htlperg
innovativ | kreativ | praxisnah

AuthentiGrade

DIPLOMARBEIT

Höhere Abteilung für Informatik

01/08/2024 – 01/04/2025

Projektmitglieder:

Christoph Amort
Klaus Mühlbachler
Thomas Fragner
Tobias Ganglberger

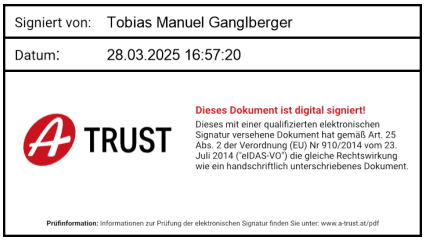
Betreuer:in:

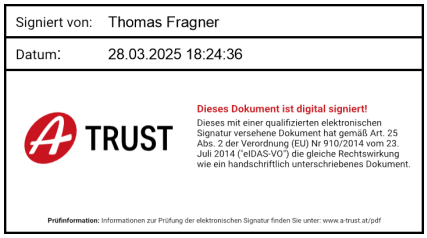
Prof. Maria Inreiter, MSc

Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

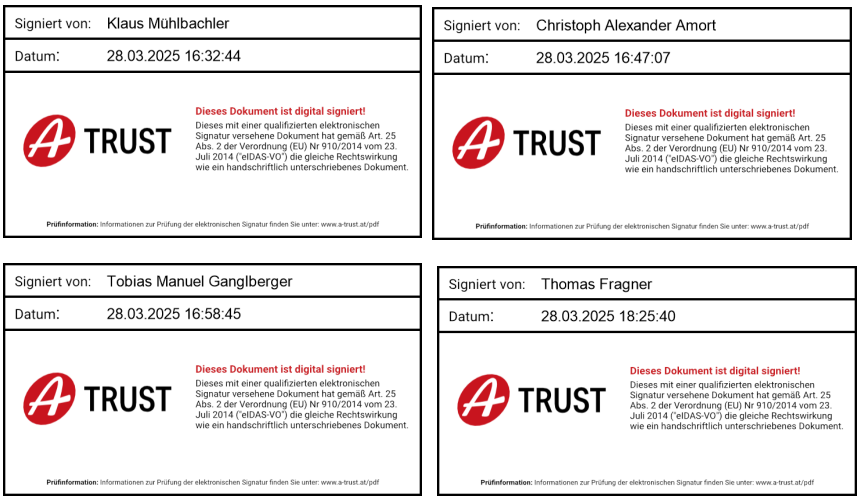
Bei der Erstellung der Arbeit haben wir keine generativen KI-Tools verwendet.





Gendererklärung

Zur besseren Lesbarkeit wird in dieser Diplomarbeit das im Deutschen übliche generische Maskulinum bei Personenbezeichnungen verwendet. Die männliche Form gilt für beide Geschlechter, es sei denn, es wird explizit anders darauf hingewiesen.



Danksagung

Wir möchten uns bei unserer Betreuerin Prof. Maria Inreiter, MSc, für ihre Untersytützung, Korrekturlesungen und Tipps während der Erstellung dieser Arbeit bedanken. Weiters bedanken wir uns bei unserem Auftraggeber, der Fabasoft Austria GmbH. Zuletzt möchten wir uns noch bei Niklas Furtlehner, welcher uns als Projekt-Betreuer von Unternehmensseite unterstützte, für die technische Unterstützung im Rahmen des Projekts bedanken.

Kurzfassung

AuthentiGrade erweitert ein bestehendes Prototyp-System um ein zentrales Authentifizierungssystem und Benutzeroberflächen für Konfigurationszwecke. Das bestehende System besteht aus mehreren Services und empfängt Daten von IoT-Geräten. Diese Daten werden in einer MongoDB-Datenbank gespeichert und anschließend ausgewertet. Zu diesen Auswertungen gehört auch das Ermitteln von Ausreißern. Außerdem sendet das bestehende System Benachrichtigungen an externe Empfänger wenn Ausreißer erkannt werden.

Für die Implementierung eines zentralen Authentifizierungssystem wird der Identity Provider Authentik aufgesetzt. Außerdem werden die bestehenden Services mit einem Authentifizierungs-Modul erweitert, welches die Services vor unberechtigten Zugriffen schützt.

Die Benutzeroberflächen für Konfigurationszwecke werden als Web-Frontends mit Angular implementiert. Außerdem werden die bestehenden Services überarbeitet, sodass sie mit den Frontend-Anwendungen kommunizieren können. Die neuen Frontends werden mit einer Library implementiert, welche es ermöglicht die Webseiten mit einem Login zu versehen, welcher über Authentik abgewickelt wird.

Diese Konfigurations-Oberflächen werden für folgende zwei Services implementiert:

- Communication-Service: Empfängt Daten von IoT-Geräten und leitet Nachrichten an externe Empfänger weiter
- Screening-Service: Wertet Daten aus der Datenbank aus und sendet Nachrichten über diese Auswertungen an den Communication-Service

Zusätzlich zu den Konfigurations-Oberflächen wird auch eine Webanwendung implementiert, welche einen direkten Einblick in die Daten bietet, welche die MongoDB-Datenbank speichert. Für diese Webanwendung wird ein Web-Frontend mittels Angular und ein Backend mittels Python implementiert.

Abstract

Authentigrade extends an existing prototype system with a central authentication system and user-interfaces for configuration purposes. The existing system consists of multiple services and receives data from IoT-Devices. The data is saved within a MongoDB-Database and then evaluated. These evaluations also include the identification of outliers. Moreover the existing system sends messages to external recipients if it detects an outlier.

The identity provider Authentik is used for the implementation of a central authentication system. In addition, the existing services are being expanded by an authentication-module, so that they are protected from unauthorized access.

The user-interfaces for configuration purposes are being implemented as web-frontends using Angular. Additionaly, the existing services are being revised, so that they can comunicate with the new frontend-applications. The new frontends are also implemented using a library, that makes it possible to provide the Websites with a login handled via Authentik.

These configuration-interfaces are being implemented for the following two existing services:

- Communication-Service: Receives from IoT-devices and forwards messages to the external recipients.
- Screening-Service: Evaluates data from the database and sends messages regarding these evaluations to the communication-service.

In addition to the configuration-interfaces, a web-application is also implemented, which offers a direct insight into the data stored by the MongoDB-database. A web-frontend using angular and a backend using python are being implemented for this web-application.

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

Dem bestehenden Projekt "Kiramet", welches in der Research Abteilung der Fabasoft AG entwickelt wird, fehlt es an Frontends und einem Authentifizierungssystem. Außerdem benötigt die Fabasoft AG eine Test-Implementierung des Identity Providers „Authentik“. Diese Test-Implementierung soll der Research Abteilung helfen, leichter Authentifizierungssysteme für kleinere Softwareprojekte zu implementieren. Infolgedessen wurde das Projekt Authentigrade umgesetzt, um das Kiramet Projekt um mehrere Frontends und ein, auf Authentik basierendes, Authentifizierungssystem zu erweitern.

1.1.1 Ausgangslage

Die Ausgangslage des Projekts besteht aus einem System von mehreren Services. Diese Services können Daten von IoT-Geräten empfangen, in einer Datenbank abspeichern und Ausreißer-Datensätze erkennen, und ein externes System über diese Ausreißer informieren. Die Services werden als Docker-Container in einer Kubernetes Umgebung gehostet.

1.1.2 Bestehendes System

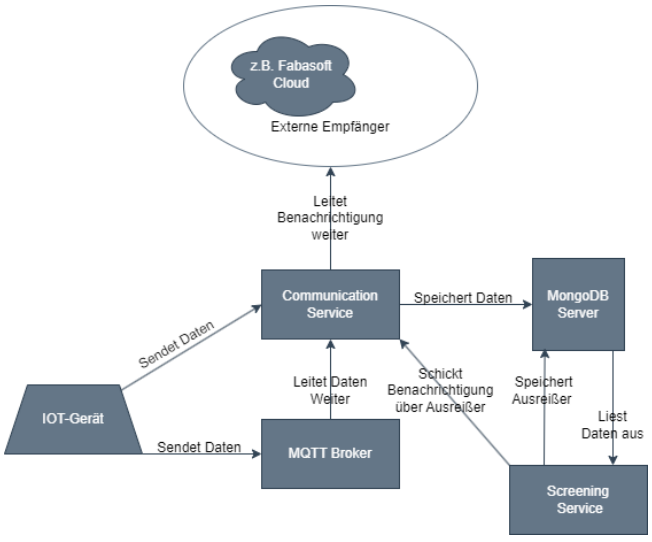


Abbildung 1: Bestehendes System

Das bestehende System, auf dem das Projekt Authentigrade aufbaut, besteht aus drei Services, welche miteinander kommunizieren, und einer Datenbank. Abbildung 1 bildet diese drei Services, die Datenbank und externe Komponenten abstrakt dar und zeigt den Kommunikationsfluss zwischen diesen.

1.1.2.1 Aufgaben der Komponenten

Der MongoDB Server ist für die Speicherung der Daten zuständig. Die folgenden drei Services übernehmen komplexere Aufgaben:

- MQTT Broker
  Der MQTT Broker empfängt Daten von einem IoT-Gerät und leitet diese an den Communication Service weiter.
- Communication Service
  Der Communication Service empfängt Daten von IoT-Geräten oder vom MQTT Broker und speichert diese in der Datenbank ab. Weiters empfängt er Benachrichtigungen über Ausreißer Daten vom Screening Service und leitet diese an einen externen Empfänger weiter. Dieser externe Empfänger ist grundsätzlich austauschbar. In der Projektumgebung in dem Authentigrade entwickelt wurde, war die Fabasoft Cloud der externe Empfänger.
- Screening Service
  Der Screening Service liest regelmäßig Datensätze aus der Datenbank aus und ermittelt Ausreißer-Datensätze. Sollte es einen Ausreißer Datensatz erkennen, wird dieser Ausreißer zusätzlich in einer Ausreißer Tabelle gespeichert. Weiters informiert der Screening Service dann den Communication Service über den Ausreißer.

1.2 Zielsetzung

Das Ziel dieser Diplomarbeit ist die Erweiterung eines Prototyp-Systems um folgende Funktionalitäten:

- Einbindung des Identity Provider Authentik
- REST-Services zur Konfiguration des Screening- und Communication-Services
- Webanwendungen zur Konfiguration des Screening- und Communication-Services
- Eine Webanwendung zur visuellen Darstellung der Daten aus der MongoDB
- Die Absicherung aller Webanwendungen durch den Identity-Provider

- Die Absicherung der Schnittstellen zur Konfiguration des Screening- und Communication-Services
- Eine zusätzliche Berechnungsmethode im Screening-Service für Tagesstatistiken von Luftfeuchtigkeitsdaten
- gleichzeitige Sicherung der Daten in mehreren Datenbanken

Die Definition und Aufgabenverteilung für die neuen Erweiterungen erfolgte durch die Nutzung von OneDev Issues. Das System soll grundlegende Funktionen beinhalten, welche in zukünftigen Systemen erweitert und produktiv verwendet werden können. Aus diesem Grund wurde besonders darauf geachtet, den geschriebenen Code qualitativ und wiederverwendbar zu gestalten, und darüber hinaus ausführlich in README-Dateien zu dokumentieren.

1.3 Projektumfeld

1.3.1 Projektteam

Das Projektteam dieser Arbeit besteht aus vier Schülern der HTL Perg: Christoph Amort, Thomas Fragner, Tobias Ganglberger und Klaus Mühlbachler.

1.3.2 Betreuung

Die Betreuung der Diplomarbeit erfolgte in enger Zusammenarbeit mit Frau Professorin Maria Inreiter, MSc., die das Projektteam von Seiten der HTL Perg unterstützte. Ergänzend dazu stellte die Fabasoft AG Herrn Niklas Furtlehner als externen Betreuer bereit. Mit seinem fundierten Fachwissen in der Webentwicklung und seiner Expertise im Fabasoft-Softwaresystem unterstützte er das Projektteam bei der erfolgreichen Umsetzung des Softwareprojekts.

1.3.3 Auftraggeber

Auftraggeber dieser Arbeit ist die Fabasoft AG, ein international agierendes Softwareunternehmen, das 1988 in Linz gegründet wurde. Seit ihrer Gründung hat sich das Unternehmen auf die digitale Dokumentenverwaltung für öffentliche Institutionen und Behörden spezialisiert.

2 Grundlagen und Methoden

2.1 Standards

2.1.1 OpenID

OpenID ist ein offenes Authentifizierungsprotokoll, das es Nutzern ermöglicht, sich mit einem einzigen digitalen Identitätsnachweis bei verschiedenen Diensten anzumelden. Dabei wird ein sogenannter „OpenID-Provider“ (z. B. Google, Microsoft oder andere vertrauenswürdige Dienste) verwendet, um die Identität des Nutzers gegenüber einem „Relying Party“-Dienst zu bestätigen [1].

OpenID trennt die Authentifizierung (also die Identitätsprüfung) von der Anwendung selbst, sodass Benutzer ihre Zugangsdaten nicht direkt beim Dienst eingeben müssen, sondern sich über den Provider identifizieren. Der Austausch erfolgt dabei in der Regel über sichere Webprotokolle wie HTTPS.

OpenID basiert auf modernen Webstandards und nutzt typischerweise das OpenID Connect Protokoll, das wiederum auf dem OAuth 2.0 Protokoll aufsetzt. OpenID Connect erweitert OAuth um Authentifizierungsfunktionen und liefert standardisierte Informationen über den angemeldeten Benutzer.

Die wichtigsten Eigenschaften von OpenID sind:

Dezentrale Authentifizierung: Nutzer können beliebige OpenID-Provider wählen. Jeder Dienst, der OpenID unterstützt, akzeptiert Authentifizierungsanfragen von verschiedenen Anbietern [1].

Wiederverwendbare Identität: Die Benutzeridentität kann auf mehreren Webseiten und Diensten verwendet werden, ohne dass für jeden Dienst neue Anmeldedaten benötigt werden [2].

Sicherheit durch Tokens: Die Authentifizierung erfolgt durch den Austausch von kryptografisch signierten Tokens, wodurch das Risiko durch Passwortdiebstahl beim Dienst reduziert wird.

Benutzerfreundlichkeit: Nutzer müssen sich weniger Passwörter merken und können sich mit einem einzigen Klick über ihren bevorzugten Anbieter anmelden

Erweiterbarkeit durch OAuth: OpenID Connect erlaubt neben der Authentifizierung auch den Zugriff auf bestimmte Benutzerinformationen, sofern der Nutzer zustimmt. Damit ist eine fein abgestufte Autorisierung möglich [3].

Standardisierte Schnittstellen: OpenID verwendet definierte Endpunkte für Authentifizierungsanfragen, Tokenaustausch und Benutzerinformationen. Diese Schnittstellen sind gut dokumentiert und breit unterstützt.

OpenID wurde ursprünglich 2005 eingeführt. Die aktuelle Version, OpenID Connect, wurde 2014 von der OpenID Foundation veröffentlicht [4].

2.1.2 REST-API

Eine REST-API (Representational State Transfer - Application Programming Interface, auch "RESTful API") ist ein Schnittstellen-Typ, welcher normalerweise HTTP verwendet. REST unterstützt mehrere Datenformate, wobei JSON am häufigsten verwendet wird[5].

Eine REST-Schnittstelle dient für den Datenaustausch zwischen einem Client und einem Server. Clients senden hierbei Anfragen mit den gängigsten HTTP-Befehlen, wie GET, PUT, POST und DELETE. Server senden dann eine Antwort mit den angefragten Informationen zurück. Damit eine API als "RESTful"gilt, muss sie sechs architektonische Bedingungen erfüllen[5]:

Einheitliche Schnittstelle: Ein Server muss Anfragen von jeder Art von REST-Client auf die gleiche Art empfangen können. Das bedeutet, dass ein REST-Client ein Browser oder eine JavaScript-App sein kann und beide auf die gleiche Art auf den Server zugreifen können.

Client-Server Architektur: Client und Server müssen explizit voneinander getrennt sein und sollen unabhängig voneinander entwickelt werden können.

Stateless: Die Anfragen vom Client an den Server sollen stateless sein, daher muss eine Anfrage alle Informationen beinhalten damit der Server die Anfrage beantworten kann. Der Server soll keine Informationen über den Client speichern um diese Information für die Beantwortung von Anfragen verwenden zu müssen.

Schichtsystem: Der Server kann aus mehreren Schichten bestehen, wobei der Client nicht wissen muss wie viele und welche Server die Anfrage bearbeiten.

Zwischenspeicher: REST-Clients sollen die Möglichkeit haben die Daten zwischenzuspeichern, um die Anfragen auf den REST-Server zu reduzieren.

Code-on-demand: Diese Bedingung ist optional und bedeutet, dass ein Server Code-Stücke an einen Client senden kann, welcher dann den Code ausführt.

REST wurde 2000 von Roy Fielding entwickelt und veröffentlicht[5].

2.2 Technologien

2.2.1 Authentik

Der Authentik Identity Provider stellt eine Open-Source-Plattform für Identitäts- und Zugriffsmanagement dar. Das System ermöglicht eine sichere Anmeldung, wobei eine einmalige Anmeldung ausreicht, um im gesamten System Zugriff auf alle verbundenen Dienste zu erhalten. Dieser Prozess wird auch als *Single Sign-On* bezeichnet. [6] [7]

2.2.2 Docker

Docker ist eine kostenlose und Quellcode-offene Software, um Anwendungen isoliert auszuführen. Dies wird unter Zuhilfenahme von Containerisierung realisiert.

2.2.2.1 Virtualisierung

Virtualisierung ist eine Technologie, um eine oder mehrere virtuelle Umgebungen auf einer einzigen physischen Maschine isoliert auszuführen. Während normale virtuelle Maschinen ein gesamtes Betriebssystem mit einem eigenen Kernel abbilden, nutzt Docker dafür Container, welche auf dem Kernel des Host-Systems aufbauen.

2.2.2.2 Container

Ein Container ist eine aktiv laufende Instanz eines Docker-Images. Ein Docker-Image enthält Anweisungen, welche ausgeführt werden, um einen Container zu starten. Jeder Container enthält seine Anwendung samt den benötigten Bibliotheken und Konfigurationen, und sie können unabhängig auf Betriebssystemen, welche Docker unterstützen, ausgeführt werden. Im Vergleich zu virtuellen Maschinen werden Container leichtgewichtig ausgeführt, was sie ressourcenschonender macht. [8] [9]

2.2.3 Kubernetes

Kubernetes ist eine Open-Source-Plattform, welche zur Verwaltung von containerisierten Anwendungen dient. Der Vorteil von Kubernetes ist, dass es eine automatisierte Skalierung, automatische Wiederherstellung von Fehlerhaften Anwendungen und effiziente Ressourcennutzung ermöglicht. Kubernetes ermöglicht es darüber hinaus, einen Dienst durch einen Load Balancer oder Ingress öffentlich freizugeben.

Kubernetes wurde 2014 von Google veröffentlicht. [10]

2.2.3.1 CronJob

Ein Bestandteil von Kubernetes ist der CronJob, welcher zeitgesteuert wiederkehrende Aufgaben ausführt. Ein CronJob erstellt bei jeder geplanten Ausführung ein neues Job-Objekt, welches die eigentliche Aufgabe ausführt, z.B. Backups oder Berichte generiert. Die Zeitplanung erfolgt in der Cron-Syntax.

Cron-Syntax Die Cron-Syntax besteht aus fünf Feldern, welche durch Leerzeichen getrennt werden. Das erste Feld steht für die Minute, das zweite für die Stunde, das dritte für den Tag des Monats, das vierte für den Monat und das fünfte für den Wochentag. Jedes dieser Felder kann entweder aus einem $*$ bestehen, also einen beliebigen Wert, oder eine Kombination aus Bereichen oder Intervallen enthalten. Beispielsweise würde `0 3 * * 1` bedeuten, dass der Cronjob jeden Montag um 3 Uhr morgens ausgeführt wird.[11]

2.2.4 Python

Python ist eine vielseitige, interpretierte Programmiersprache, die sich durch einfache Syntax und umfangreiche Standardbibliotheken auszeichnet. Sie wird in vielen Bereichen eingesetzt, darunter Webentwicklung, Datenanalyse, Künstliche Intelligenz und Automatisierung [12].

Die Sprache ermöglicht eine schnelle Entwicklung durch dynamische Typisierung und eine große Auswahl an Drittanbieter-Bibliotheken [13]. Python folgt dem Prinzip der Lesbarkeit und fördert eine saubere, verständliche Code-Struktur [14].

Folgendes Beispiel zeigt, wie eine einfache Funktion in Python definiert und aufgerufen wird:

```python
# Funktion definieren
def greet(name):
    return f"Hallo, {name}!"
```

```
# Funktion aufrufen
print(greet("Welt"))
```

2.2.4.1 FastAPI

FastAPI ist ein modernes, leistungsfähiges Python-Webframework, das für die Erstellung von APIs entwickelt wurde. Es basiert auf Starlette für Webanfragen und Pydantic für Datenvalidierung und bietet automatische OpenAPI- und Swagger-Dokumentation [15].

Das Framework ermöglicht es, API-Endpunkte mit Python-Funktionen zu definieren, die automatisch die Anfragen verarbeiten und Antworten zurückgeben [16]. FastAPI unterstützt asynchrone Programmierung, was eine effiziente Verarbeitung von Anfragen ermöglicht [17]. Der folgende Code zeigt, wie eine einfache API mit einem GET-Endpunkt erstellt wird:

```
from fastapi import FastAPI

# FastAPI-Instanz erstellen
app = FastAPI()

# Einfacher GET-Endpunkt
@app.get("/")
async def read_root():
    return {"message": "Hello, World!"}
```

2.2.4.2 PyMongo

PyMongo ist eine offizielle Python-Bibliothek, die es ermöglicht, mit einer MongoDB-Datenbank zu interagieren. Sie bietet eine API, um Datenbanken zu erstellen, Daten zu lesen, aktualisieren und löschen [18].

Die Bibliothek ermöglicht es eine Verbindung zu einer MongoDB-Datenbank in einem Objekt zu speichern und dann mit einem Schlüssel-basiertem Zugriff auf die Sub-Datenbanken und Collections der Datenbank zuzugreifen [19] [20]. Durch das Zugreifen auf die Collections ist es dann möglich Dokumente einzufügen, auszulesen, zu löschen und zu bearbeiten. Folgender Code zeigt wie das Verbinden zur Datenbank und das Zugreifen auf eine Sub-Datenbank und Collection erfolgt:

```
# Verbindung aufbauen
client = MongoClient("mongodb://localhost:27017/")

# Schlüsselbasierter Zugriff auf Sub-Datenbank
db = client["Sub-Datenbank-Name"]

# Schlüsselbasierter Zugriff auf Collection
collection = db["Collection-Name"]
```

2.2.4.3 requests

Requests ist eine Python-Bibliothek, welche genutzt wird, um einfach und unkompliziert HTTP/1.1-Anfragen zu senden. Die Bibliothek stellt verschiedene Tools zur Verfügung, um die HTTP-Anfragen anzupassen und entsprechende Antworten zu erhalten. Sie gehört nicht zur Standardinstallation von Python, auch, wenn sie bei den Benutzern zu den beliebtesten gehört. [21]

Der ursprüngliche Author von Requests ist Kenneth Reitz und wurde erstmals am 14. Februar 2011 veröffentlicht. [22]

2.2.4.4 httpx

HTTPX ist ein HTTP-Client für Python, welcher sowohl synchrone als auch asynchrone Anfragen ermöglicht. Sie ist eine moderne Alternative zu requests und zeichnet sich durch ihre bessere Performance aus. Die Bibliothek unterstützt darüber hinaus auch noch zahlreiche weitere Features, welche Requests nicht unterstützt, darunter beispielsweise:

- Asynchrone Kompatibilität
- HTTP/2 Protokoll

[23] [24]

2.2.4.5 uvicorn

Uvicorn ist ein leistungsstarker, asynchroner Webserver für Python, der speziell für ASGI-Anwendungen entwickelt wurde. Er ermöglicht eine schnelle und effiziente Verarbeitung von Webanfragen und wird oft in Kombination mit FastAPI oder Starlette verwendet [25].

Der Server nutzt eine ereignisgesteuerte Architektur und basiert auf `uvloop` und `httptools`, wodurch er im Vergleich zu traditionellen WSGI-Servern eine hohe Performance bietet [26]. Das folgende Code-Beispiel zeigt, wie eine FastAPI-Anwendung mit Uvicorn gestartet wird:

```python
from fastapi import FastAPI
import uvicorn

# FastAPI-Instanz erstellen
app = FastAPI()

# Einfacher GET-Endpunkt
@app.get("/")
async def read_root():
    return {"message": "Hello, World!"}

# Server mit Uvicorn starten
if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

2.2.4.6 bson

Bson (Binary Javascript Object Notation) ist eine von MongoDB unabhängige Python-Bibliothek [27], welche allerdings hauptsächlich in Kombination mit MongoDB verwendet wird. MongoDB nutzt nämlich hauptsächlich das BSON-Format, um dessen Daten zu organisieren und zu speichern. Im Vergleich zum herkömmlichen JSON unterstützt BSON auch Datentypen für das Datum und binäre Daten. [28]

2.2.4.7 datetime

Die offizielle Python-Bibliothek `datetime` wird genutzt, um mit dem Datum und der Zeit zu arbeiten. Dazu stellt sie Klassen bereit, darunter zählen folgende zu den gängigsten:

- date: Dies ist das klassische Datum, also Tag, Monat und Jahr
- time: Dies ist ein vom Datum unabhängiger Zeitpunkt, welcher die Stunden, Minuten, Sekunden und Millisekunden enthält
- datetime: Hierbei handelt es sich um eine Kombination aus dem Datum und der Zeit

- timedelta: Diese Klasse wird genutzt, um die Differenz zwischen zwei Zeitpunkten zu berechnen

[29]

2.2.4.8 kubernetes

`kubernetes` ist eine Python-Bibliothek, welche die Kommunikation mit der Kubernetes-API übernimmt, um mit Kubernetes zu interagieren. Sie stellt eine Reihe von Modulen, darunter Client und Config, bereit. Config wird genutzt, um die Kubernetes-Cluster-Konfigurationen zu laden, und Client, um mit den Ressourcen aus dem Kubernetes-Cluster zu interagieren. Der Client stellt dazu verschiedene API-Klassen zur Verfügung, darunter beispielsweise **CoreV1API** [30], um mit den Kern-Ressourcen von Kubernetes zu interagieren, und **BatchV1API** [31] zur Verwaltung von Jobs und Cronjobs. Zur Erleichterung der Nutzung übernimmt die Bibliothek außerdem auch Aufgaben wie beispielsweise die Authentizierung bei der API. [32]

2.2.4.9 uuid

`uuid` ist eine Python Bibliothek, welche verwendet wird, um UUIDs zu generieren. Eine UUID (Universally Unique Identifier) ist eine 128-Bit-Zahl, welche genutzt wird, Objekte in Computersystemen zu identifizieren. Zur Generierung der UUID werden verschiedene Quellen, beispielsweise die Zeit oder die Geräte-MAC-Adresse, genutzt. Die Bibliothek stellt verschiedene Versionen, um UUIDs zu generieren, bereit, darunter:

- UUID Version 1: Diese Version nimmt den Zeitpunkt und die MAC-Adresse des Geräts als Basis, um die UUID zu generieren
- UUID Version 3: Bei dieser Version werden eine Namespace-UUID und ein angegebener Name kombiniert, und anschließend mit MD5-Hashing berechnet, um die UUID zu generieren
- UUID Version 4: Bei dieser Version werden zufällige Zahlen genommen, um die UUID zu generieren. Es wird sich nicht auf Details wie den Zeitpunkt oder eine MAC-Adresse verlassen.

[33]

2.2.4.10 base64

Die Python-Bibliothek `base64` wird verwendet, um Binärdaten in das Base64-Format zu kodieren und auch wieder zu dekodieren. [34]

2.2.4.11 pydantic

Pydantic ist eine weit verbreitete Python Bibliothek, welche genutzt wird, um Daten zu validieren und zu serialisieren.

```python
from pydantic import BaseModel


class Person(BaseModel):
    first_name: str
    last_name: str


validating = Person(first_name="John", last_name="Doe")
```

[35] Eines der beliebtesten Wege, ein Pydantic-Schema zu definieren, ist die Nutzung von Model-Klassen, welche Attribute mit ihren zugehörigen Regeln enthält. Beispielsweise kann man dann das oben gezeigte Model nutzen, um zu überprüfen, ob sich ein mitgegebener Body bei einer HTTP-Request im richtigen Format befindet.

2.2.4.12 json

JSON (Java Script Object Notation) ist ein Datenformat, welches genutzt wird, um Daten strukturiert zwischen Systemen auszutauschen. Der Vorteil von JSON gegenüber anderen Datenformaten ist, dass es nicht an Plattformen oder Programmiersprachen gebunden, leichtgewichtig und lesbar ist [36]. Die Python-Bibliothek `json` ermöglicht es nun, JSON-Strings in Python-Objekte umzuwandeln und umgekehrt. [37]

2.2.4.13 pytest

pytest ist ein Open-Source Test-Framework für Python, welches für das schnelle Entwickeln von automatisierten Tests geeignet ist. Es erkennt Test-Dateien automatisch wenn sie *test_* als Namenspräfix haben und nutzt es Python' Standard-assert-Anweisungen. Außerdem ist es durch eine Vielzahl von Plugins erweiterbar[38].

pytest wurde von Holger Krekel entwickelt und erstmals im Jahr 2004 veröffentlicht[39].

2.2.5 MongoDB

MongoDB (siehe Logo in Abbildung 2) ist ein nicht-relationales Datenbank Management System, welches Daten in Form von Dokumenten speichert, welche ein JSON-Objekt repräsentieren. MongoDB ist optimiert für das schnelle Speichern und Abfragen großer Datenmengen und eine einfach Skalierbarkeit[41]. Der Name MongoDB leitet sich auch vom englischen Begriff "Humongous"(Deutsch: Gigantisch) ab[42].

Abbildung 2: MongoDB Logo [40]

MongoDB ist in 3 Versionen verfügbar [41]:

MongoDB Community: Kostenlose Open-Source Version von MongoDB

MongoDB Enterprise: Kostenpflichtige Version mit zusätzlichem Support und Sicherheits-Features.

MongoDB Atlas: Cloud-Basierte Version von MongoDB. Kostenlos verfügbar als kleine Test-Version, sonst Kostenpflichtig.

MongoDB Community und Enterprise laufen auf allen gängigen Betriebssystemen. Im Rahmen dieses Software-Projekts wird es als Docker-Container gehostet.

Ein MongoDB-Server speichert mehrere Sub-Datenbanken (auch "Databases"), welche wiederum mehrere Collections speichern. Eine Collection kann eine unbegrenzte Menge an Dokumenten speichern und kann mit Tabellen aus relationalen Datenbanken verglichen werden. Die Dokumente in einer Collection müssen keine vorgegebene Struktur besitzen und können daher unterschiedliche Attribute speichern. Es ist aber möglich Collections mit sogenannten Schemas zu konfigurieren, so dass Dokumente gewisse Attribute besitzen müssen bzw. gewisse Attribute einen gewissen Datentyp speichern müssen [43].

MongoDB besitzt eine eigene Query-Language, welche in Aggregations-Operationen und CRUD-Operationen aufgeteilt ist. Die CRUD-Operationen sind als Funktionsaufrufe[44] implementiert, während die Aggregations-Operationen durch Funktionsaufrufe und JSON-Objekte ausgeführt werden, welche Äggregation Pipelines"genannt werden[45]. MongoDB unterstützt außerdem Programmiersprachen-basierten Zugriff für alle bekannten Programmiersprachen, wie Python, Java oder Typescript.

MongoDB ist 2009 von 10gen (seit 2013 "MongoDB Inc.") veröffentlicht worden, wobei die Entwicklung bereits 2007 begonnen hat [42].

2.2.6 Angular

Angular ist ein von Google entwickeltes Framework für die Erstellung von Single Page Applications (SPAs). Angular basiert auf TypeScript, einer Weiterentwicklung von Javascript, und bietet eine neue Struktur für die Entwicklung von Webanwendungen. Durch die komponentenbasierte Architektur Angulars wird eine saubere Trennung von Logik und Darstellung gewährleistet. [47]

Abbildung 3: Angular Logo [46]

2.2.6.1 Komponenten

Eine Angular-Applikation besteht aus Komponenten. Eine Komponente setzt sich aus drei grundlegenden Bestandteilen zusammen:

- **HTML:** In der HTML-Datei befindet sich der HTML-Code einer Komponente.
- **SCSS:** SCSS wird zur Definition des Stylings einer Komponente verwendet. Als Weiterentwicklung von CSS bietet SCSS erweiterte Funktionen, die eine strukturierte und übersichtliche Gestaltung von Stylesheets ermöglichen.
- **Typescript:** In einer Angular-Komponente steuert TypeScript die Logik. Es definiert die Komponente als Klasse, verwaltet Daten, verarbeitet Benutzerinteraktionen und bindet dynamische Inhalte ins HTML-Template ein. Durch statische Typisierung und moderne Features sorgt TypeScript für eine strukturierte und fehlerarme Entwicklung.

[48]

2.2.6.2 Single Page Applications

Eine Single Page Application ist eine Webanwendung, die ohne vollständige Neuladevorgänge auskommt. Statt ganze Seiten vom Server zu laden, werden nur benötigte Daten nachgeladen und der Inhalt dynamisch aktualisiert. Dies führt zu einer schnellen, nahtlosen Benutzererfahrung, ähnlich einer Desktop-Anwendung. Angular eignet sich besonders für SPAs, da es durch Komponenten, Routing und State-Management eine effiziente Struktur bietet. [49]

2.2.6.3 Node.js

Node.js ist eine serverseitige Laufzeitumgebung für JavaScript, die auf der V8-Engine basiert. Sie ermöglicht die Ausführung von JavaScript außerhalb des Browsers und wird häufig für Backend-

Entwicklung eingesetzt. In Angular-Projekten wird Node.js benötigt, um das Angular-CLI auszuführen, Abhängigkeiten über den Paketmanager NPM zu verwalten und den Entwicklungsserver bereitzustellen. [50]

2.2.6.4 NPM

NPM ist der Paketmanager für Node.js und verwaltet Abhängigkeiten in Angular-Projekten. Mit NPM können Bibliotheken installiert, aktualisiert und verwaltet werden. Zudem enthält es Skripte zur Automatisierung von Entwicklungsaufgaben, wie das Starten des Angular-Servers oder das Bauen der Anwendung. [51]

2.2.6.5 Typescript

TypeScript ist eine auf JavaScript basierende Programmiersprache mit statischer Typisierung. In Angular wird TypeScript verwendet, um Komponenten, Services und andere Teile der Anwendung strukturiert und typsicher zu entwickeln. Es bietet moderne Features wie Interfaces, Klassen und Module, die die Wartung und Skalierbarkeit des Codes verbessern. [52]

2.2.6.6 Angular-oauth2-oidc

Das Paket angular-oauth2-oidc ermöglicht die Integration von OAuth2 und OpenID Connect (OIDC) in Angular-Anwendungen. Es unterstützt Authentifizierung, Autorisierung sowie Token-Verwaltung und erleichtert die sichere Kommunikation mit APIs. Typische Funktionen sind das Speichern von Zugriffstokens, das automatische Erneuern von Tokens und die Verwaltung von Benutzerinformationen. In dieser Arbeit wird dieses Paket verwendet, um die Benutzer mithilfe von Authentik zu identifizieren. [53]

2.2.7 nginx

Nginx ist ein leistungsstarker und Open-Source Webserver. Neben dem Hosten von Webseiten kann Nginx auch als Reverse-Proxy, Load Balancer und HTTP-Cache verwendet werden [54]. Im Rahmen dieses Software-Projekts wird Nginx in Form von Docker Containern verwendet um Webseiten zu hosten.

Nginx wurde von Igor Sysoev entwickelt und am 4. Oktober 2004 veröffentlicht[55].

2.3 Tools

2.3.1 OneDev

OneDev ist ein Open-Source Git-Server, den man selbst hosten kann, um Repositories zu verwalten. Neben der Repositoryverwaltung unterstützt OneDev außerdem auch CI/CD, welches man über eine Benutzeroberfläche intuitiv erstellen und verwalten kann. Darüber hinaus kann man in OneDev auch Issues erstellen und diese in Kanban-Boards strukturieren, um eine effiziente Entwicklung in einem Team zu gewährleisten. Ein weiterer Vorteil von OneDev ist die Geschwindigkeit und ressourcenschonende Nutzung. [56]

2.3.2 Rancher

Rancher ist eine Open-Source-Plattform zur Verwaltung von Kubernetes-Clustern. Sie ermöglicht es, mehrere Kubernetes-Cluster zentral zu verwalten, unabhängig davon, ob sie lokal, in der Cloud oder auf hybriden Infrastrukturen betrieben werden. Rancher bietet eine benutzerfreundliche Oberfläche, um Cluster bereitzustellen, zu konfigurieren und zu überwachen [57].

Darüber hinaus integriert Rancher Funktionen wie rollenbasierte Zugriffskontrolle (RBAC), Monitoring, Logging und CI/CD. Ein wesentlicher Vorteil ist die einfache Verwaltung von Kubernetes-Ressourcen ohne tiefgehende Kenntnisse der Kubernetes-CLI, was besonders in Teams die Zusammenarbeit und Produktivität fördert [58].

2.3.3 Coder

Coder ist eine selbst hostbare Open-Source Plattform, um Cloud-Entwicklungsumgebungen zu erstellen und zu verwalten. Diese Entwicklungsumgebungen stellen die Infrastruktur, IDE's und Werkzeuge bereit, welche benötigt werden, um den Code zu schreiben. [59] Die Programmierung erfolgt nicht lokal auf dem Rechner, sondern in der Cloud, was in einer Einsparung der Ressourcen des lokalen Rechners resultiert. Mit Coder ist es zudem möglich, Vorlagen der Workspaces mit anderen Teammitgliedern zu teilen. [60] Dies hat den Vorteil, dass man nicht alle mit der Entwicklung verbundenen Werkzeuge und Abhängigkeiten lokal installieren muss, wodurch potentielle Inkonsistenzen vermieden werden.

Coder wurde im August 2017 von Ammar Bandukwala, Kyle Carberry und John Andrew Entwistle geschaffen. [61]

3 Konzept

Das Konzept Kapitel beschreibt aus welchen Komponenten das Ergebnis des Projekts besteht und wie sie miteinander interagieren.

3.1 Überblick

Das Projekt erweitert das bestehende System um mehrere Bestandteile. Das System soll unter anderem um Authentik, dem Identity Provider, erweitert werden. Es sind zwei Benutzeroberflächen für die Konfiguration des Communication- und Screening-Service geplant. Weiteres sollen bestehende Services erweitert werden, um neue Funktionalität in den Benutzeroberflächen zu ermöglichen. Außerdem ist eine Web-Anwendung, inklusive Backend, für das Auslesen von Daten aus der Datenbank geplant. Außerdem werden Module, für eine vereinfachte Kommunikation zwischen den Services und Authentik und den Kubernetes-Files, implementiert.

3.1.1 Beispiel-Ablauf im fertigen System

Ein IoT-Gerät sendet Sensordaten an einen MQTT-Broker, welcher die Daten im Anschluss wiederum an einen REST-Endpoint des Communication-Service sendet. Alternativ kann das IoT-Gerät die Daten auch direkt an den Communication-Service senden. Die Sensordaten werden im folgenden Format versendet:

```
{
    id: int
    timestamp: str
    description: str
    value: float
    unit: str
}
```

Die erhaltenen Daten werden im Anschluss vom Communication-Service in einer MongoDB-Datenbank persistiert, und können dann in der MongoDB-UI betrachtet werden. Der Screening-Service ruft in regelmäßigen Abständen die Sensor-Daten aus der Datenbank ab und führt entsprechende Berechnungen mit ihnen durch. Diese Berechnungen werden dann zurück in die Datenbank gespeichert. Zusätzlich schickt der Screening-Service Nachrichten über die Berechnungen an den Communication-Service. Am Schluss sendet der Communication-Service diese Nachrichten an einen externen Empfänger.

Alle Benutzeroberflächen und REST-Endpoints werden dabei durch Authentik mit der Open-ID-Authentifizierungsschicht geschützt. Die Benutzeroberflächen werden geschützt, indem ein Login benötigt wird, um auf diese zuzugreifen. Die Backends werden geschützt, indem ein Token benötigt wird, um Anfragen an diese zu schicken. In Abbildung 4 ist der Datenfluss der Sensordaten und Berechnungen im fertigen System grafisch dargestellt.

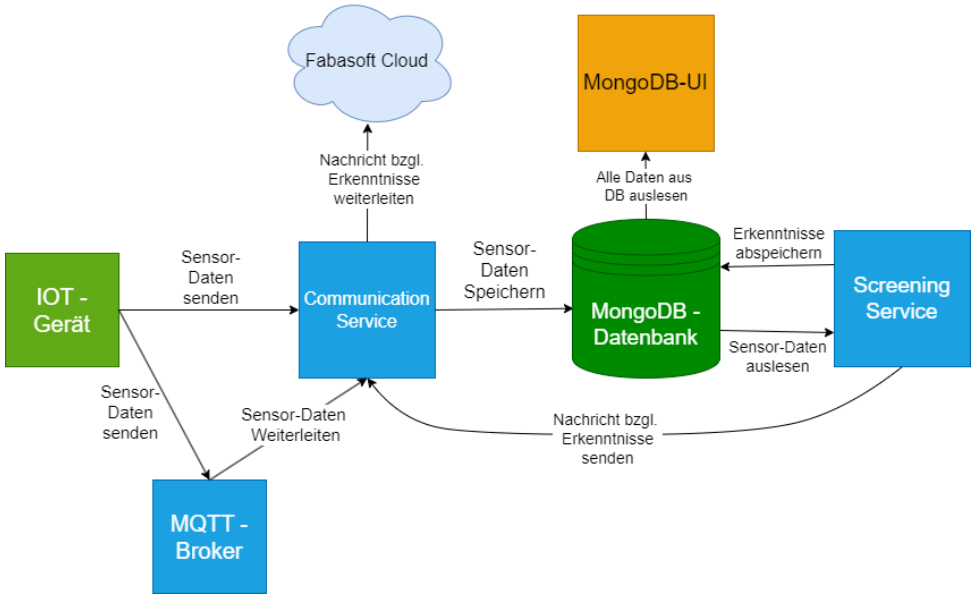


Abbildung 4: Datenfluss im fertigen System

3.2 Bestehendes System

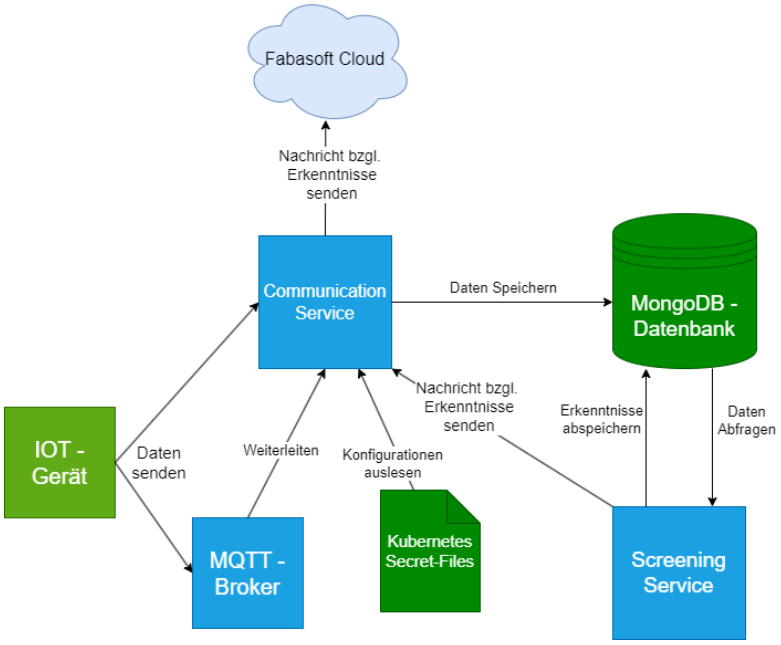


Abbildung 5: Bestehendes System

Die in Abbildung 5 dargestellte Grafik beschreibt die Struktur des bestehenden Systems, welches unser Projekt schrittweise erweitert.

3.3 Screening Service UI

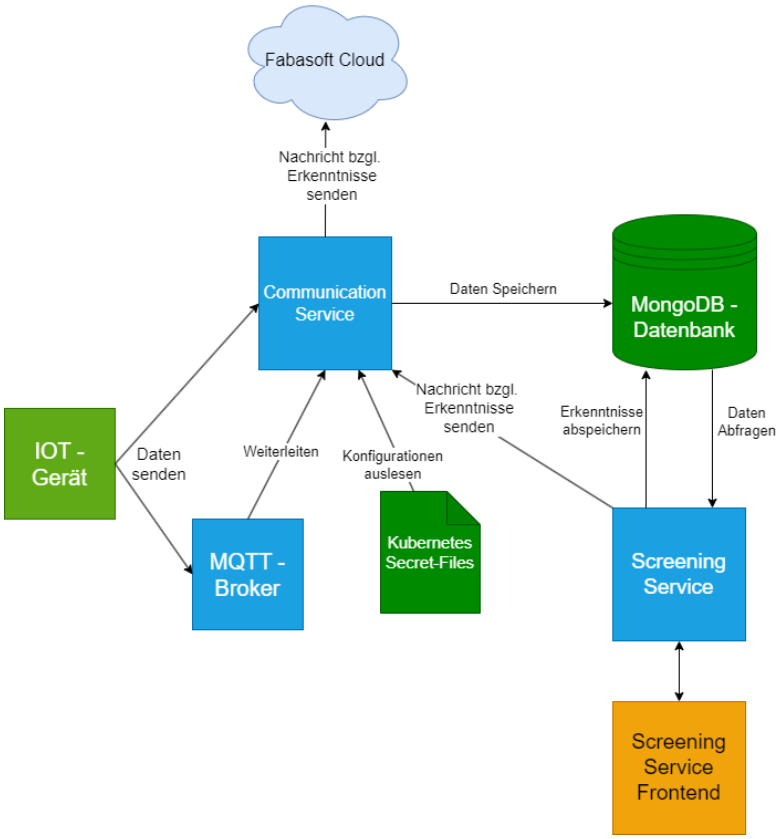


Abbildung 6: Screening Service UI

Das UI des Screeningservices wird, wie in Abbildung 6 zu sehen ist, in das System eingebunden.

Das UI des Screening-Service ist als Angular Frontend geplant und dient dazu, den Screening-Service zu konfigurieren. Die Kommunikation zwischen Screening-Service und Benutzeroberfläche ist mittels REST-API in Planung. Im UI sind folgende Einstellungen und Operationen geplant:

- Berechnungsintervall
- Berechnungsmethoden: Es soll möglich sein, bestimmte Screening-Service-Berechnungen über eine Checkbox ein- oder auszuschalten.
- Communication-Service-URLs: Im UI soll es möglich sein, anzupassen, an welche Communication-Services die Ergebnisse der Berechnungen gesendet werden.
- Sofortige Ausführung der Screening-Service-Berechnungen

3.4 Screening Service Erweiterung

Der Screening-Service wurde um eine zusätzliche Berechnungsmethode erweitert. Der Screening-Service empfängt nun auch regelmäßig Daten über die Luftfeuchtigkeit, und berechnet basierend auf diesen Daten einfache Statistiken wie den Durchschnitt, den Mindest- und Höchstwert sowie die Standardabweichung eines Tages. Diese Berechnungen werden in einer Datenbank persistiert, und eine Nachricht mit den Ergebnissen wird an den Communication-Service weitergeleitet. Des Weiteren wurden die in Abschnitt **3.3** erwähnten Konfigurationsmöglichkeiten als REST-Endpoints implementiert, sowie ein weiterer Endpoint zum abrufen folgender Informationen:

- Communication-Service-URLs
- Berechnungsintervalle
- alle Cronjobs in einem Kubernetes-Namespace
- alle Screening-Service Berechnungsmethoden inklusive Status, ob die Methode aktiviert ist oder nicht.

3.5 Communication Service UI

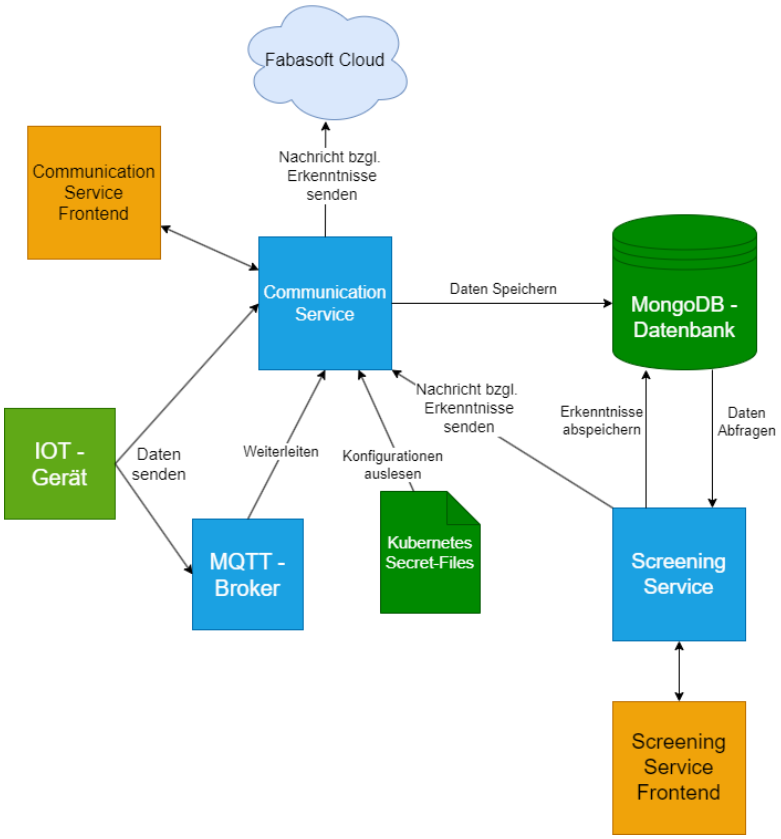


Abbildung 7: Communication Service System

In Abbildung 7 ist zu sehen, wie das UI in das System eingebunden wird.

Die Communication Service-UI ist als Weboberfläche geplant, die mit Angular implementiert wird. Sie dient der Konfiguration des Communication Services.

Die Kommunikation zwischen Communication-Service und Backend wird über eine REST-API erfolgen. In der Weboberfläche soll folgendes eingestellt werden können:

- Nachrichten Weiterleitungen
- URLs, die zur Weiterleitung verwendet werden.
- Datenbankverbindungen hinzufügen oder löschen

3.6 Communication Service Erweiterung

Der Communication Service wurde um fünf API-Endpoints erweitert und die bestehenden wurden mit einer Authentifizierung erweitert.

3.6.0.1 API-Endpoints

Die API stellt sieben Endpoints zur Verfügung, welche alle auf GET-, POST-, PUT- und DELETE-Requests hören:

3.6.1 Bestehende-Endpoints

3.6.1.1 POST: /CAQ

Dieser Endpoint empfängt Daten in Form eines `InformationMessage`-Objekts und leitet diese an aktive Service-URLs weiter. Dabei wird überprüft, ob die jeweilige Verbindung aktiv ist und die Daten korrekt verarbeitet werden können. Im Fehlerfall, etwa bei ungültigen Daten oder fehlgeschlagener Verbindung, wird eine Fehlermeldung zurückgegeben. Erfolgreiche Übertragungen werden mit einer entsprechenden Bestätigung quittiert.

3.6.1.2 POST: /sensor-data

Dieser Endpoint empfängt Sensordaten in Form eines `SensorData`-Objekts. Die übermittelten Daten werden in der MongoDB-Datenbank gespeichert. Nach erfolgreicher Speicherung wird eine Bestätigungsmeldung zurückgegeben.

3.6.2 Neue-Endpoints

3.6.2.1 GET: /comm-service-urls

Dieser Endpoint gibt eine Liste aller verfügbaren Service-URLs aus. Falls ein Fehler auftritt, etwa durch ungültige Eingaben oder einen internen Verarbeitungsfehler, wird eine entsprechende Fehlermeldung zurückgegeben.

3.6.2.2 GET: /comm-service-url

Dieser Endpoint gibt die URL eines spezifischen Kommunikationsservices zurück. Die Abfrage erfolgt anhand eines eindeutigen Namens (`name`). Falls die angeforderte URL nicht existiert oder ein Fehler auftritt, wird eine Fehlermeldung zurückgegeben.

3.6.2.3 POST: /comm-service-url

Dieser Endpoint erlaubt es, eine neue Kommunikations-Service-URL hinzuzufügen oder eine bestehende zu aktualisieren. Erfolgreiche Änderungen werden mit einer Bestätigungsmeldung bestätigt.

3.6.2.4 DELETE: /comm-service-url

Dieser Endpoint ermöglicht das Löschen einer bestimmten Kommunikations-Service-URL anhand ihres Namens (`name`). Nach erfolgreicher Löschung wird eine Bestätigung zurückgegeben.

3.6.2.5 PUT: /comm-service-url/{name}

Dieser Endpoint erlaubt es, den Status (aktiv/inaktiv) einer bestehenden Kommunikations-Service-URL zu aktualisieren. Nach erfolgreicher Aktualisierung wird eine Bestätigung zurückgegeben.

3.7 Secret-Communication-Module

Das Secret-Communication-Module besteht aus mehreren Funktionen und dient der Verwaltung von Kubernetes-Secrets, die Kommunikationsdienst-URLs enthalten. Es ermöglicht das Abrufen, Aktualisieren, Hinzufügen und Löschen von Service-URLs, die in den Kubernetes-Secrets gespeichert sind. Das Modul ist zuständig für die sichere Bearbeitung der Konfiguration von Kommunikationsdiensten und verwendet hierfür die Kubernetes-API.

Wie man in der Abbildung 12 sieht, wurden die Kubernetes-Secret-Files aktualisiert.

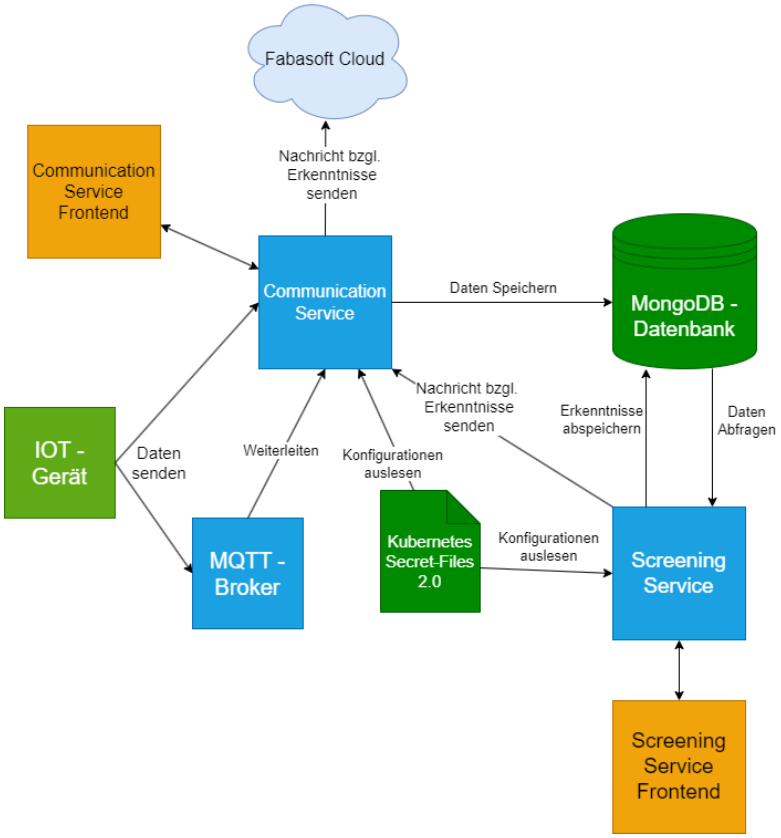


Abbildung 8: Systemstruktur des Secret-Communication-Modules

3.8 Database-Module

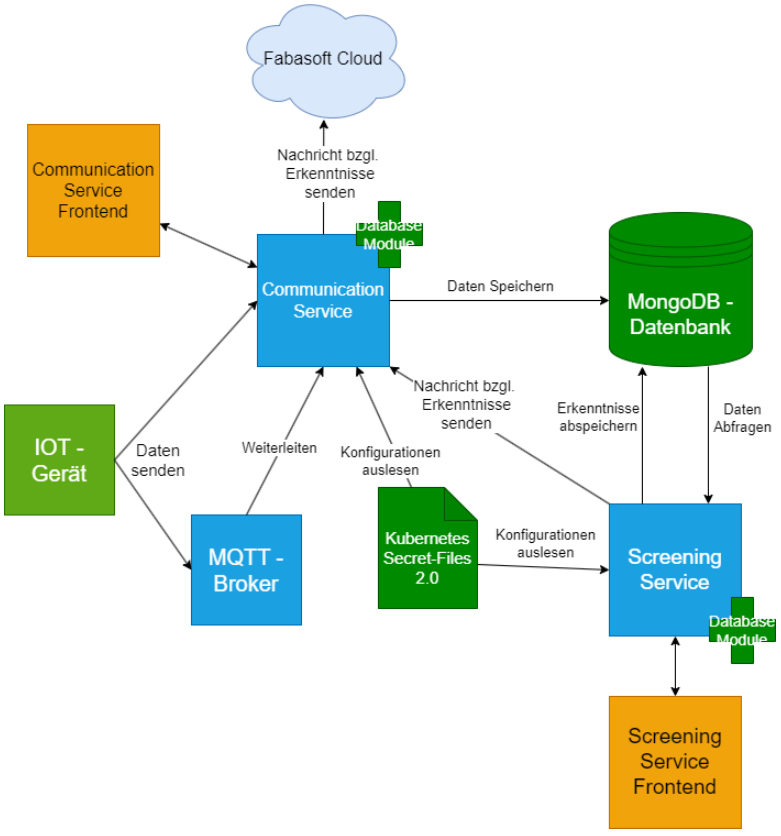


Abbildung 9: Systemstruktur: Database-Module-Erweiterung

Das Database-Module stellt einen API-Router mit Endpoints bereit, um MongoDB-URLs abrufen, löschen und hinzufügen zu können. Diese MongoDB-URLs werden sowohl vom Screening- als auch vom Communication-Service genutzt, um Daten zu speichern. Dazu stellt das Module zum einen eine Funktion bereit, welche das Speichern auf mehreren MongoDB-Instanzen ermöglicht, und zum anderen den bereits erwähnten API-Router, welcher im Screening- und Communication-Service-Backend, wie in Abbildung 9 ersichtlich, eingebunden wird. Da die MongoDB-URLs ähnlich wie die in Abschnitt **??** erwähnten Kommunikationsdienst-URLs gespeichert werden, verwenden die Router-Endpoints Funktionen des Secret-Communication-Modules, um die MongoDB-Urls in den Kubernetes-Secrets zu verwalten. Die Konfigurationsoberflächen des Screening- und Communication-Services nutzen schlussendlich diese Endpoints, um die MongoDB-URLs zu verwalten.

3.9 MongoDB UI

Die MongoDB UI ist eine Webanwendung welche es ermöglicht alle Daten aus der MongoDB Datenbank auszulesen. Sie besteht aus einem Frontend und einem Backend und kommuniziert mit der Datenbank (siehe Abbildung 6). Sowohl das Frontend als auch das Backend kommunizieren über die jeweiligen Libraries und Module mit Authentik für die Authentifizierungsverfahren.

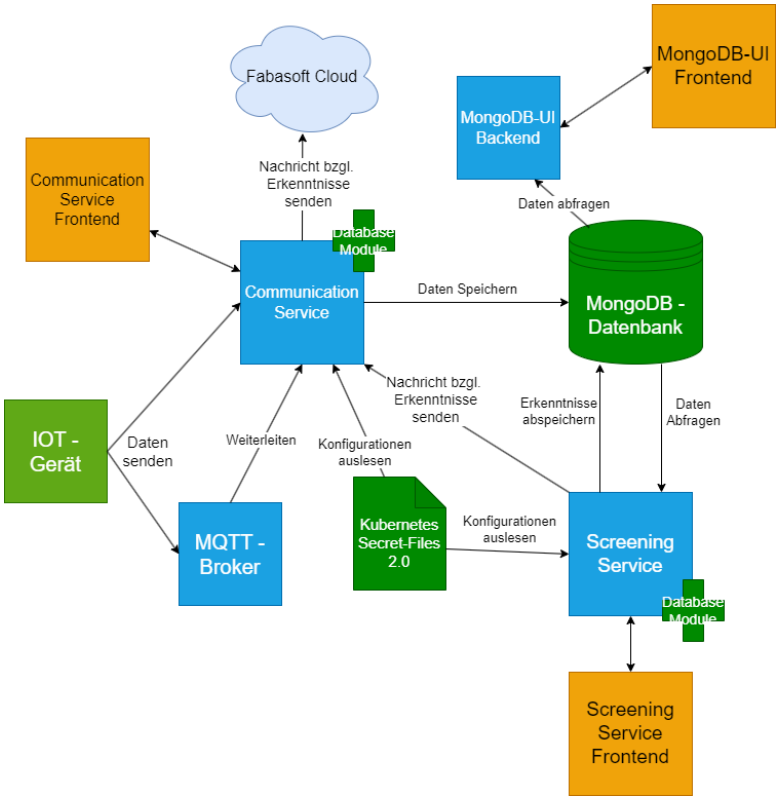


Abbildung 10: MongoDB-UI im System

3.9.1 Backend

Das Backend ist als Python-Applikation, welche Daten aus der Datenbank über eine REST-API zur Verfügung stellt, geplant.

3.9.1.1 API-Endpoints

Es sind drei REST-API Endpoints geplant, welche alle auf GET-Requests hören. Diese Endpoints sollen folgende Aufgaben übernehmen:

- Bereitstellen einer Liste der Namen aller Sub-Datenbanken in der MongoDB-Datenbank
- Bereitstellen einer Liste der Namen aller Collections in einer Sub-Datenbank
- Bereitstellen einer Liste aller Dokumente einer Collection

3.9.1.2 Datenbank-Zugriff

Das Backend soll Daten aus der Datenbank auslesen aber keine Daten bearbeiten.

3.9.2 Frontend

Für die Entwicklung des Frontends ist die Verwendung des Angular-Frameworks geplant. Es soll möglich sein die Sub-Datenbank und Collection, aus welchen die Dokumente angezeigt werden, auszuwählen. Außerdem sind Filter- und Sortier-Funktionalitäten vorgesehen.

3.9.2.1 Daten erhalten

Es ist vorgesehen, dass das Frontend die Daten erhält, indem es REST-API-Anfragen an das Backend sendet, welches dann die angefragten Daten zurücksendet.

3.9.3 Nutzer-Anwendungsfall

Ein Nutzer kann über das Frontend einsehen, ob neue Daten von einem Sensor an den Communication-Service geschickt wurden. So kann er überprüfen ob der Sensor noch läuft und regelmäßig Daten sendet. Ein Nutzer kann die UI auch verwenden um die Einträge in den Analyse-Collections direkt einzusehen. Auch für einen direkten Einblick in die Outliers-Collection kann die UI verwendet werden. So kann ein User selbst ermitteln ob ein Sensor ein auffälliges Verhalten zeigt und oft Ausreißer-Daten sendet bzw. gar keine Daten mehr schickt. Anhand dieser Informationen kann der Nutzer dann

Fehlersuche betreiben. In Abbildung 11 ist ein Use-Case Diagramm zu diesem Nutzer-Anwendungsfall zu sehen.

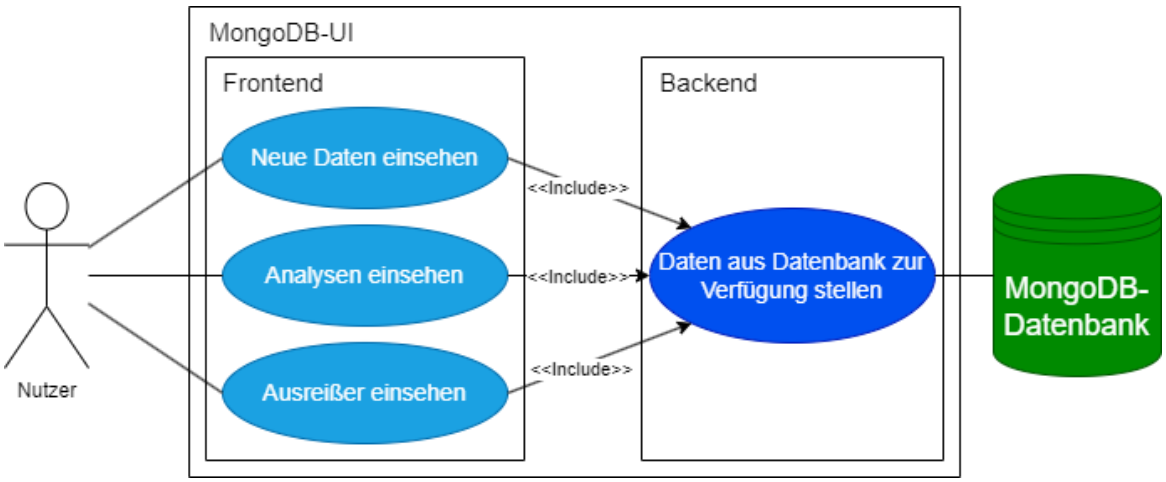


Abbildung 11: Use-Case Diagramm für die MongoDB-UI

3.10 Identity Provider

Der Identity-Provider Authentik besteht aus mehreren Docker Containern. Authentik ist als Identity Provider geplant und somit für die Speicherung aller Benutzer und Berechtigungen verantwortlich. Zusätzlich sollen Rollen mit unterschiedlichen Rechten angelegt werden können, die dann den Nutzern zugewiesen werden können sollen. Die Authentik-Komponente übernimmt die Authentifizierung der Nutzer.

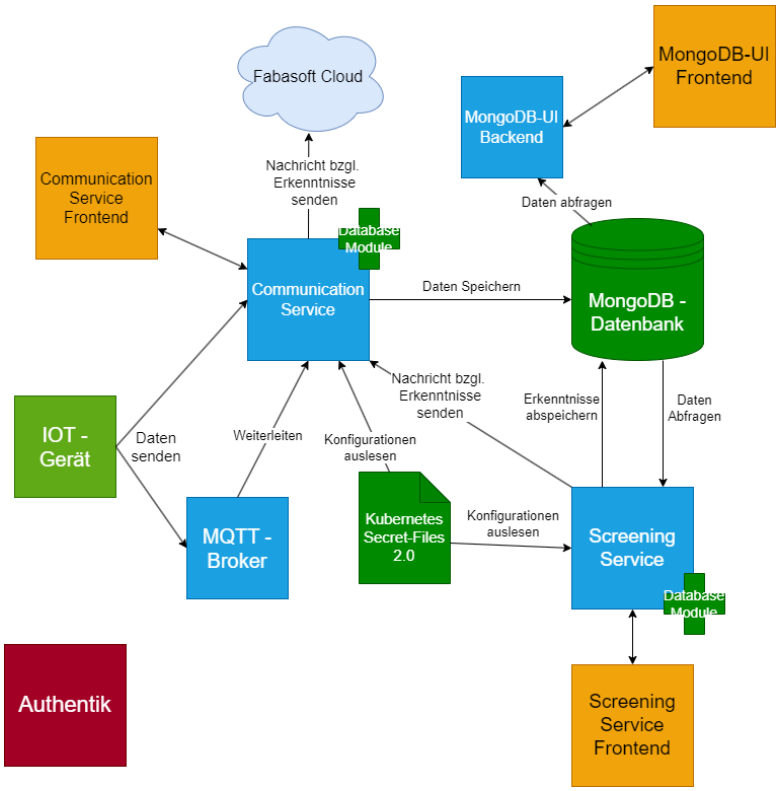


Abbildung 12: Authentik im System

3.11 Frontend-Authentifizierung

Alle Frontends verwenden das Angular Framework. Daher wird für alle Frontends die selbe Bibliothek verwendet um eine Kommunikation mit Authentik zu ermöglichen. Diese Bibliothek ist angular-oauth2-oidc. Sie leitet Nutzer, welche nicht eingeloggt sind, automatisch zu Authentik weiter. Sie übernimmt auch die Kommunikation mit Authentik um die Tokens zu verwalten und zu überprüfen. Die Kommunikation mit Authentik und welche Frontends abgesichert werden wird in Abbildung 13 grafisch dargestellt.

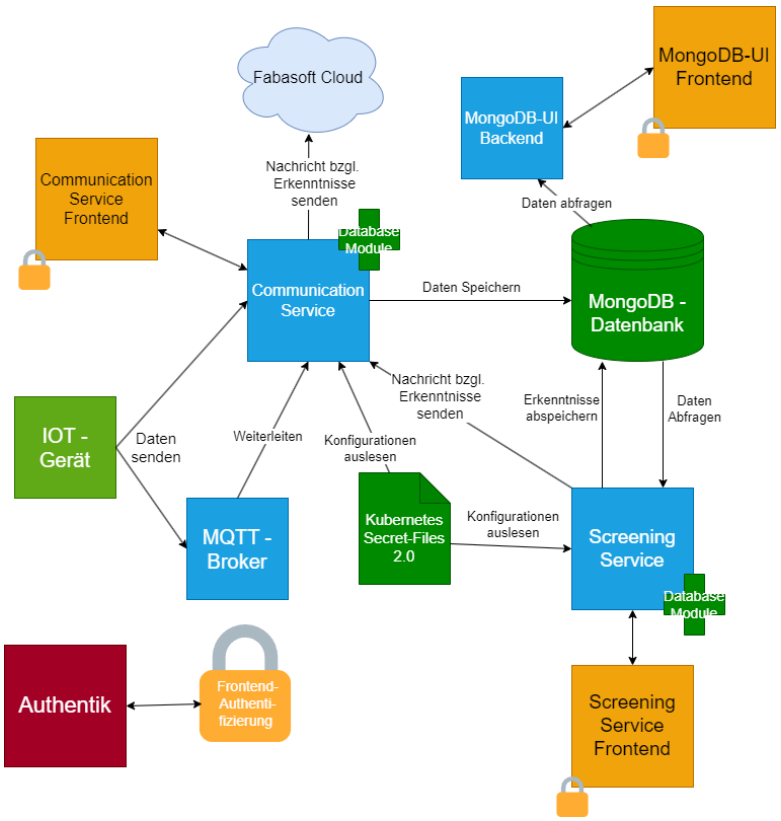


Abbildung 13: Systemstruktur der Frontend Authentifizierung

3.12 Authentication-Module

3.12.1 Funktion

Das AuthenticationModule besteht aus mehreren Modulen und dient der Authentifizierung und Autorisierung von Nutzern und Services. Es basiert auf dem OpenID Connect (OIDC)-Standard und integriert sich mit einem Identity-Provider wie Authentik. Das AuthenticationModule ist für das Identitätsmanagement zuständig und stellt sicher, dass Zugriffstoken validiert und Nutzer eindeutig identifiziert werden können. Zusätzlich können öffentliche Schlüssel (JWKS) vom Identity-Provider abgerufen werden, um die Token-Verifizierung durchzuführen. Die Authentifizierungslogik unterstützt verschiedene Scopes und erlaubt die Verwendung von Service-Accounts für maschinelle Kommunikation.

3.12.2 Verwendung

Das AuthenticationModule wird zur Authentifizierung der API-Endpoints verwendet. Für den Zugriff auf diese Endpoints ist ein gültiger Token erforderlich. Wird kein Token mitgesendet, antwortet das System mit dem HTTP-Statuscode 401 *Unauthorized*", der standardmäßig bei fehlgeschlagener

Authentifizierung zurückgegeben wird. Der Token muss im HTTP-Header enthalten sein und folgendem Format entsprechen:

Die Struktur der Authentifizierung im Backend ist in Abbildung 14 dargestellt.

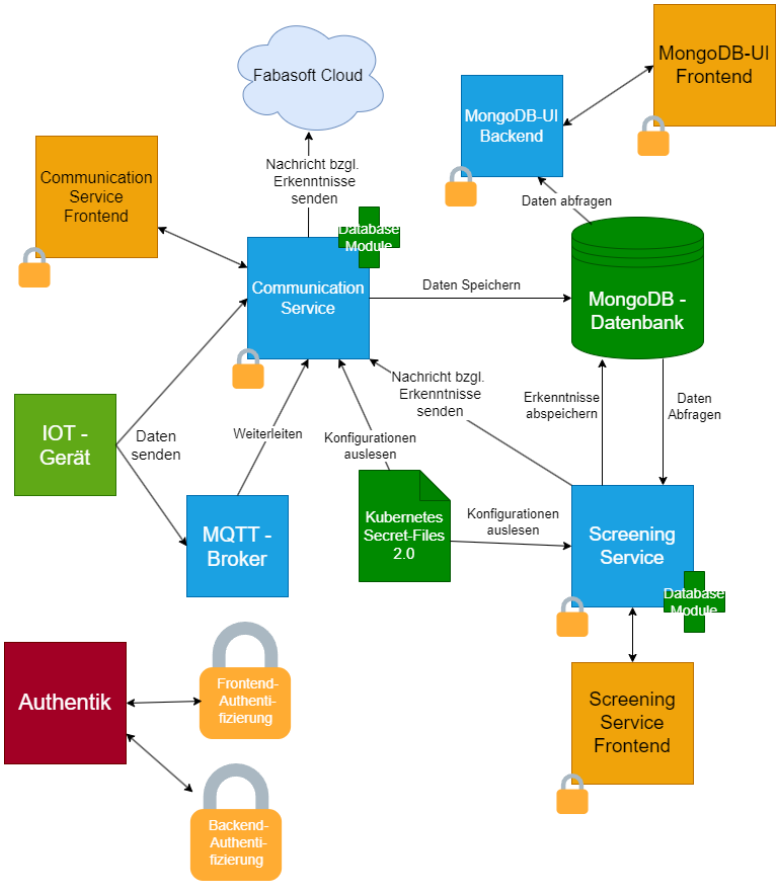


Abbildung 14: Systemstruktur des AuthenticationModules im Backend

4 Implementierung

4.1 Deployment

Alle Services und Webseiten, die im Rahmen dieser Diplomarbeit entwickelt wurden, werden über eine automatisierte CI/CD-Pipeline mit OneDev als CI/CD-Tool deployed. Die Anwendung wird dabei als Docker-Container gebaut, versioniert und anschließend auf ein Kubernetes-Cluster ausgerollt.

4.1.1 CI/CD

In diesem Abschnitt wird der Ablauf eines Deployments mittels CI/CD beschrieben. Dabei wird sowohl auf die Buildspec-Konfiguration in OneDev als auch auf die Kubernetes-Deployment-Datei (`k8s.yml`) eingegangen. Der Fokus liegt darauf, den Weg vom Code bis hin zur laufenden Applikation im Cluster nachvollziehbar darzustellen [62].

4.1.1.1 OneDev Buildspec

Die OneDev Buildspec-Datei legt fest, welche Schritte bei einem Push ins Repository automatisch durchgeführt werden [63].
Typische Schritte sind:

- Der Build des Docker-Images
- Das Pushen des Images in die Container-Registry
- Das anschließende Deployment ins Kubernetes-Cluster

Ein beispielhafter Auszug aus einer `.onedev-buildspec.yml`-Datei:

```
jobs:
  - name: Build and Deploy
    steps:
      - run: docker build -t
       ↪ harbor.res.fabagl.fabasoft.com/zero3/comm_service_api:bjoern .
```

```
      - run: docker push
      ↪  harbor.res.fabagl.fabasoft.com/zero3/comm_service_api:bjoern
      - run: kubectl apply -f k8s.yml
```

Hierbei ist insbesondere der Image-Name:
`harbor.res.fabagl.fabasoft.com/zero3/comm_service_api:bjoern`
relevant, welcher sowohl im Buildprozess als auch in der Kubernetes-Definition verwendet wird.

4.1.1.2 Kubernetes File

Die Datei `k8s.yml` beschreibt die Ressourcen, die im Kubernetes-Cluster für die Anwendung angelegt werden sollen. In der vorliegenden Konfiguration werden zwei Ressourcen definiert: ein `Deployment` und ein `Service` [64].

Ein Ausschnitt der verwendeten Datei sieht folgendermaßen aus:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: commservice-api-caq-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: api
  template:
    metadata:
      labels:
        app: api
    spec:
      containers:
        - name: commservice-api-caq
          image: harbor.res.fabagl.fabasoft.com/zero3/comm_service_api:bjoern
          ports:
            - containerPort: 8000
---
apiVersion: v1
```

```
kind: Service
metadata:
  name: commservice-api-caq-svc
spec:
  ports:
    - port: 8000
      targetPort: 8000
  selector:
    app: api
```

Wichtige Punkte in diesem YAML sind:

- **Name des Deployments:** `commservice-api-caq-deployment`
- **Name des Services:** `commservice-api-caq-svc`
- **Docker-Image:** `harbor.res.fabagl.fabasoft.com/zero3/comm_service_api:bjoern`, welches direkt aus dem CI/CD-Prozess stammt
- **Expose Port:** 8000 über Deployment und Service

Diese Definitionen sorgen dafür, dass die Anwendung konsistent, automatisiert und reproduzierbar im Cluster bereitgestellt wird.

4.1.2 Dockerfiles

Jede Anwendung im System besteht aus einem oder mehreren Dockerfiles. Die Dockerfiles für die Frontend und Backend Anwendung sind jeweils sehr ähnlich aufgebaut. In den folgenden Unterkapiteln wird der grundlegende Aufbau und Inhalt der Dockerfiles der Backend und Frontend Anwendungen erläutert.

4.1.2.1 Backend

Für die Docker-Container der Backend-Services werden standardmäßig folgende Build-Schritte in den Dockerfiles angegeben:

- Als Basis-Image wird standardmäßig python-slim oder python-slim-bullseye verwendet.
- Nach der Angabe des Basis-Image werden die Umgebungsvariablen für das Image angegeben. Das beinhaltet Umgebungsvariablen, welche für alle Images notwendig sind, wie die Proxy-

Einstellungen und die Verbindungsdaten für den Identity Provider. Hier werden aber auch service-spezifische Umgebungsvariablen angegeben.

- Nach dem Definieren der Umgebungsvariablen wird im Container in das jeweilige Arbeitsverzeichnis gewechselt. Dieses kann unter /app oder /srv liegen.
- Dann folgt der Abschnitt wo der gesamte Code und die requirements.txt, welche alle benötigten Libraries beinhaltet, sowie alle externen Module, die im CI/CD Prozess ins Verzeichnis geladen wurden, in den Container kopiert werden.
- Basierend auf dem letzten Schritt werden dann die Bibliotheken aus der requirements.txt Datei installiert.
- Zuletzt wird der Service gestartet, hierfür wird entweder ein Python-Befehl oder Uvicorn-Befehl ausgeführt.

Ein Muster-Dockerfile für einen Backend-Service befindet sich im Anhang C.1.

4.1.2.2 Frontend

Die Dockerfiles für die Frontend-Anwendungen erstellen immer zwei Container. Der erste ist temporär und nur für den Build-Prozess der Webanwendung, der zweite Container ist ein nginx-Webserver, der die fertige Webseite hostet.

Für den Build-Container werden standardmäßig folgende Build-Schritte im Dockerfile definiert:

- Als Basis-Image wird ein einfaches node Image verwendet. Außerdem wird der Container bei der Basis-Image-Definition mittels *AS build* als Build-Container markiert, was für spätere Zwecke relevant ist.
- Im Build-Container werden in den Umgebungsvariablen nur die Proxy-Konfigurationen festgelegt. Danach wird das Arbeitsverzeichnis im Container festgelegt, welches sich standardmäßig unter /app befindet.
- Im nächsten Schritt wird die package.json Datei in den Container kopiert. Anschließend werden die Befehle *npm install* und *npm install -g @angular/cli* ausgeführt. Diese Befehle installieren alle benötigten Bibliotheken.
- Der vorletzte Schritt im Build-Container besteht darin, den Code in das Arbeitsverzeichnis des Containers zu kopieren.
- Zuletzt wird der Befehl *ng build –configuration=production* ausgeführt, was den Code zu einer Webseite zusammenbaut.

Für den Hosting-Container werden standardmäßig folgende Build-Schritte im Dockerfile definiert:

- Als Basis-Image wird ein nginx-Image verwendet.
- Im Hosting-Container werden nach der Angabe des Basis-Image die restlichen Umgebungsvariablen definiert. Diese beinhalten die Verbindungsdaten für den Identity Provider und die Anwendungs-spezifischen Umgebungsvariablen.
- Dann werden die Webseiten-Dateien die im Build-Container generiert wurden, aus dem Build-Container in den Hosting-Container kopiert. Hierfür wird beim Verzeichnispfad des Build-Containers als Präfix *–from=build* angegeben.
- Zusätzlich zu jedem Frontend-Dockerfile gibt es eine entrypoint.sh Datei, diese wird im nächsten Schritt in den Hosting-Container kopiert.
- Die entrypoint.sh Datei wird im letzten Schritt ausgeführt. Die .sh-Datei führt für jede Umgebungsvariable, die im Hosting-Container gesetzt wurde, einen Shell-Befehl aus, welcher die Umgebungsvariablen in der main.js-Datei, die der Build-Container generiert hat, an der richtigen Stelle platziert. Dafür wird der *sed* Shell-Befehl genutzt, der den Platzhalter-Wert, der in einer *environment.ts* Datei angegeben wurde, sucht und mit der richtigen Umgebungsvariable ersetzt. Ein solcher *sed*-Befehl kann wie folgt aussehen:

  ```
  #          Platzhalter Wert | Umgebungsvariable | Pfad zur main.js Datei
  sed -i "s|http://127.0.0.1:3000/|$API_URL|g" /nginx/html/main-*.js
  ```

- Am Ende der entrypoint.sh-Datei wird der Befehl *nginx -g daemon off;* ausgeführt, was den Webserver startet.

Ein Dockerfile für eine Frontend-Anwendung befindet sich im Anhang C.2.

4.2 Authentik

Authentik fungiert als Identity Provider sowohl für Frontend- als auch für Backend-Systeme. Die Authentik-Instanz wurde mithilfe des Paketmanagers HELM in einem Kubernetes-Cluster bereitgestellt.[6]

4.2.1 Aufsetzen in Kubernetes

Die Authentik-Instanz dieser Arbeit wurde in einem Kubernetes-Cluster bereitgestellt. Für die Installation und Verwaltung von Authentik innerhalb des Clusters wurde der Paketmanager Helm verwendet. Mit der Datei VALUES.YAML werden die benötigten Container für eine Authentik Instanz definiert. Der Kubernetes Cluster besteht im Wesentlichen aus vier Docker Containern. Darunter eine Redis Datenbank, eine Postgresql Datenbank, Authentik selbst und ein NGINX Server.

values.yaml Datei

```yaml
authentik:
    secret_key: "SECRET_KEY" #Verwendung für Kryptografie und
    ↪ Sitzungsverwaltung
server:
  ingress:
    ingressClassName: nginx #Authentik wird über NGINX erreichbar
    enabled: true
    hosts:
        - "HOST_NAME (AUTH.EXAMPLE.COM)"
    tls: #Zugriff über HTTPS
      - hosts: [ "HOST_NAME (AUTH.EXAMPLE.COM)" ]
        secretName: tls-authentik #Zertifikat gespeichert unter
  env:
  - name: HTTP_PROXY #Fabasoft spezifische Proxyeinstellungen
    value: "--"
redis: #Caching-Backend. Z.B. für Sitzungsspeicherung
    enabled: true
```

Authentik mit Helm installieren Die Bereitstellung von Authentik innerhalb eines Kubernetes-Clusters erfolgt mithilfe des Paketmanagers HELM.

Um die Authentik-Instanz im Kubernetes-Cluster aufzusetzen, werden die folgenden Befehle ausgeführt:

```
helm repo add authentik https://charts.goauthentik.io
helm repo update
helm upgrade --install authentik authentik/authentik -f values.yaml
```

Diese Befehle bewirken zunächst die Einbindung des offiziellen Authentik-Helm-Chart-Repositories in die Helm-Umgebung. Anschließend wird das Repository aktualisiert, um sicherzustellen, dass stets die neuesten Chart-Versionen zur Verfügung stehen. Abschließend erfolgt die Installation bzw. Aktualisierung von Authentik im Kubernetes-Cluster unter Berücksichtigung der in der Datei `values.yaml` spezifizierten Konfigurationsparameter.

Durch die Verwendung von HELM wird der Installationsprozess erheblich vereinfacht, da manuelle Konfigurationsschritte und die Erstellung individueller Kubernetes-Manifeste entfallen. Dies trägt zur Reproduzierbarkeit und Wartbarkeit der Infrastruktur bei.

Danach kann die Authentik Instanz über die vorher in der 'values.yaml' Datei angegebene Domain erreicht werden. Bei dem ersten Aufruf der Instanz muss ein Passwort für den Standard Administrator „akadmin“ festgelegt werden, mit dem dann auf das Admin-Interface zugegriffen werden kann.

4.2.2 Authentik User Interface

Authentik stellt sowohl für Administratoren als auch für Benutzer eine Benutzeroberfläche bereit, über die sämtliche Dienste angezeigt und verwaltet werden können. Administratoren haben zudem die Möglichkeit, Einstellungen anzupassen und administrative Aufgaben durchzuführen.

4.2.2.1 Dashboard

Das Dashboard dient dem Benutzer als Portal, in dem alle Dienste, die mit Authentik authentifiziert werden, angezeigt und aufgelistet werden. Dies ermöglicht eine schnelle Übersicht auf alle verfügbaren Dienste. Alle verfügbaren Dienste können in Gruppen zusammengefasst werden, wodurch das Dashboard eine klar strukturierte Benutzeroberfläche ist. In Abbildung 15 ist das Dashboard eines Benutzers ohne Administrator Rechte zu sehen.

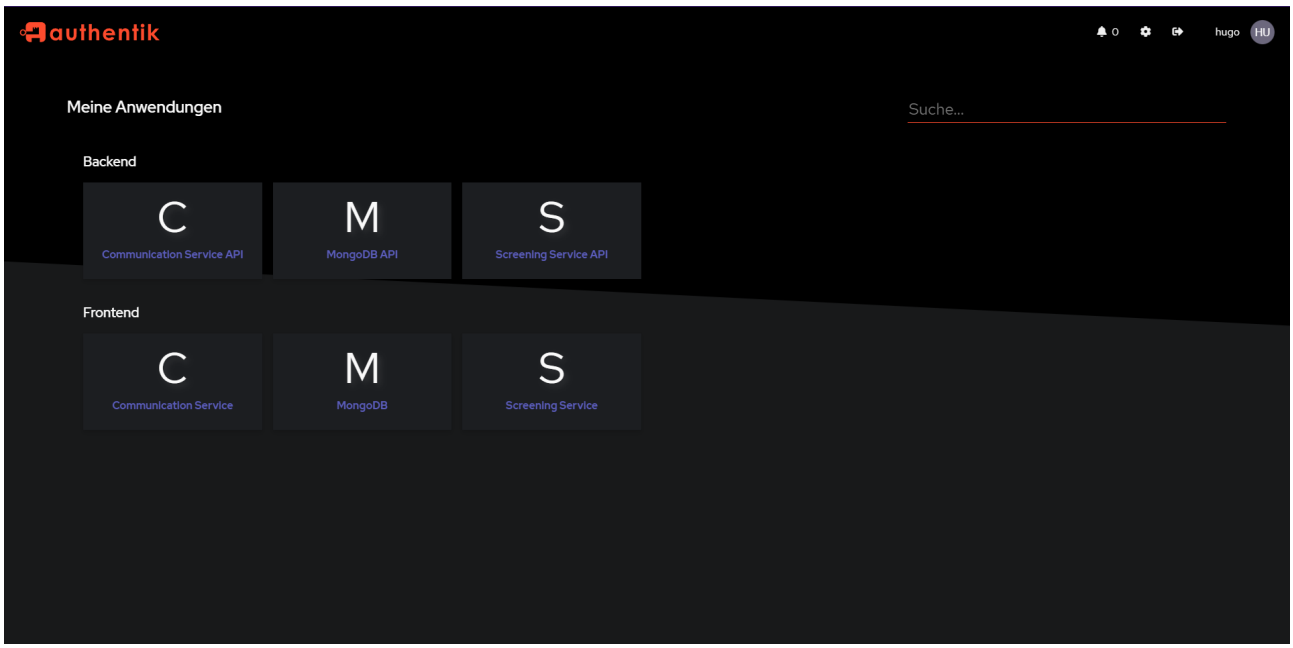


Abbildung 15: Authentik Dashboard

4.2.2.2 Admin Interface

Der Administrator hat über einen speziellen Zugang die Möglichkeit, das Administratorinterface zu öffnen. Dieser Zugang ist über einen deutlich gekennzeichneten Button im Dashboard erreichbar, der den Titel „Admin Interface“ trägt. Der Button ist exklusiv für Administratoren sichtbar, sodass nur berechtigte Nutzer darauf zugreifen können. Durch diesen gezielten Zugang wird sichergestellt, dass administrative Funktionen und Einstellungen nur von autorisierten Personen eingesehen und bearbeitet werden können.

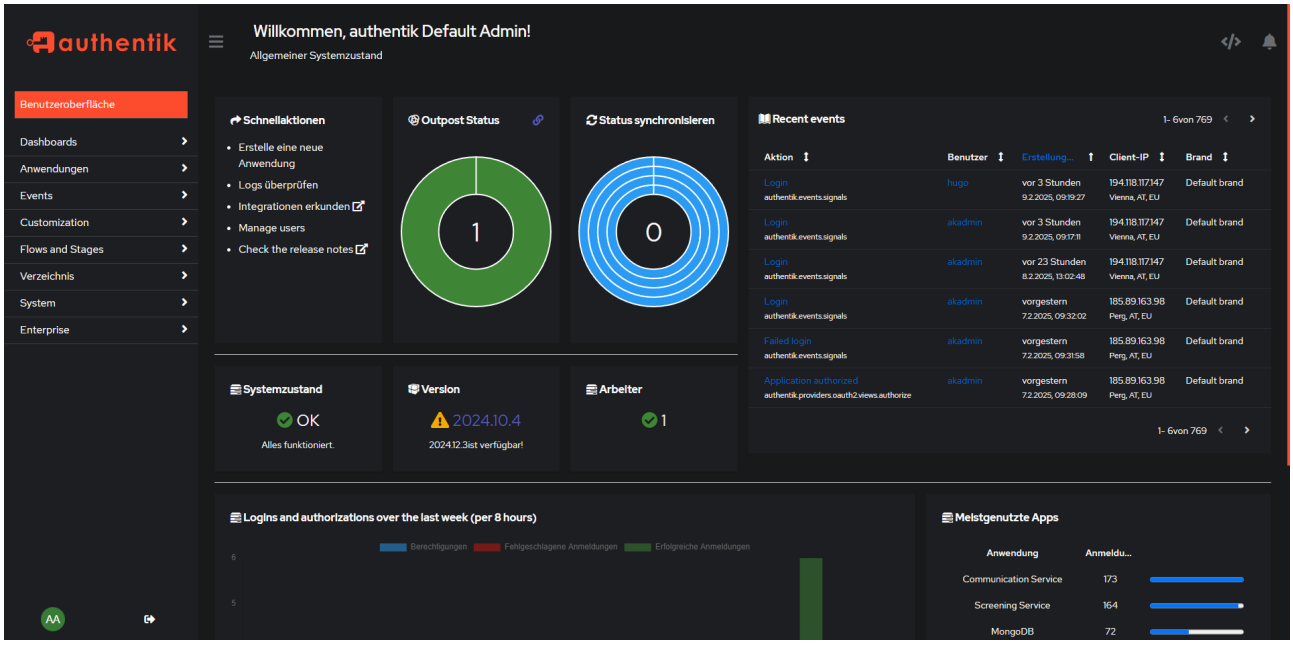


Abbildung 16: Authentik Admin Interface

Im Admin-Interface hat der Administrator die Möglichkeit, die gesamte Authentik Instanz zu verwalten und anzupassen. Wie in Abbildung 16 zusehen ist, befindet sich auf der linken Seite des Interfaces eine Liste, die alle verschiedenen Einstellungs- und Verwaltungsmöglichkeiten enthält. Diese Liste ist in Kategorien unterteilt wie zum Beispiel: „Anwendungen“ oder „Events“, was das Admininterface leicht verständlich und klar strukturiert macht. Zusätzlich zu diesen Einstellungsmöglichkeiten kann der Administrator relevante Daten der Instanz einsehen. Dazu gehören beispielsweise aktuelle Sitzungen angemeldeter Nutzer sowie der Betriebsstatus der Instanz, einschließlich möglicher Fehlermeldungen oder Systemstörungen.

4.2.3 Rechteverwaltung

Authentik bietet Administratoren eine umfassende und flexible Berechtigungsverwaltung, um den Zugriff auf verschiedene Anwendungen und Ressourcen zu steuern. Benutzer können Gruppen zugeordnet werden, wodurch sich deren Rechte zentral verwalten lassen. Gruppen können wiederum Rollen erhalten, die eine Sammlung spezifischer Berechtigungen enthalten. Alternativ können Rollen auch direkt Benutzern zugewiesen werden. Auf diese Weise lässt sich der Zugriff gezielt regeln, sodass jeder Benutzer nur auf die für ihn vorgesehenen Anwendungen und Ressourcen zugreifen kann. Diese strukturierte Zugriffskontrolle trägt wesentlich zur Sicherheit des Systems bei.

4.2.3.1 Rollen

Mithilfe von Rollen können in Authentik Berechtigungen verwaltet werden. Diese Rollen werden im Admin Interface unter dem Menüpunkt „Verzeichnis –> Rollen“ verwaltet, wo sie erstellt, bearbeitet oder gelöscht werden können.

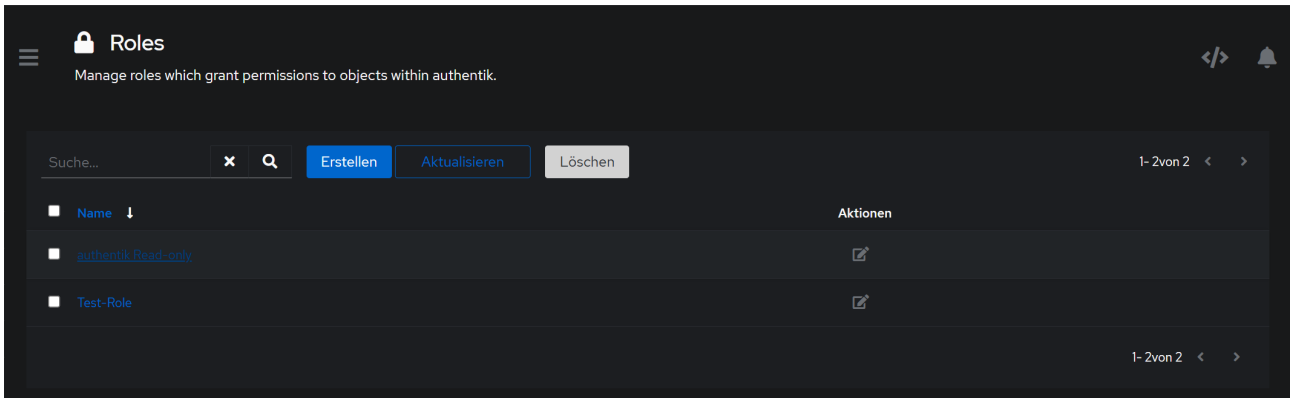


Abbildung 17: Authentik Rollen Übersicht

Erstellt man eine neue Rolle im Admininterface wird man zunächst dazu aufgefordert einen Namen für die neue Rolle einzugeben. Danach ist die neu angelegte Rolle im Admininterface in der Rollen-

verwaltung zu sehen. Um sie nun zu bearbeiten, drückt man auf den Namen der Rolle und man gelangt in die Detailansicht. In dieser Detailansicht gibt es zwei Menüpunkte am oberen Rand. Für die Berechtigungsverwaltung ist lediglich der Menüpunkt „Permissions“ relevant.

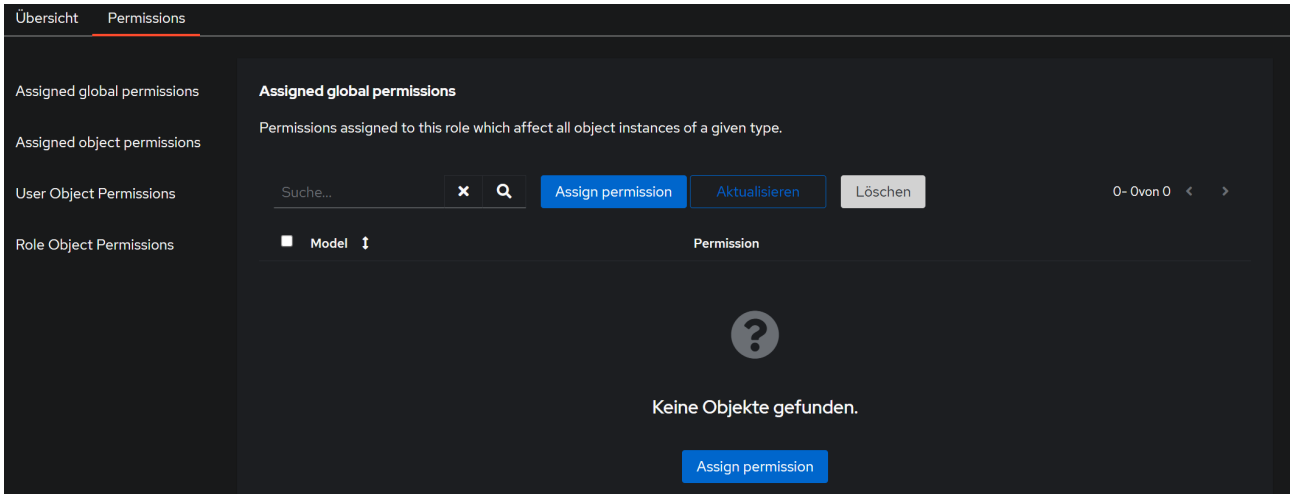


Abbildung 18: Authentik Rollen Verwaltung

Wie in der Abbildung 18 zu sehen ist, gibt es am linken Rand der Rollenverwaltung vier Menüpunkte, die sich jeweils auf unterschiedliche Berechtigungen beziehen.

Assigned Global Permissions: In diesem Menüpunkt können der Rolle Berechtigungen zugewiesen werden, die sich auf die gesamte Authentik-Instanz beziehen. Diese Berechtigungen gelten systemweit und sind nicht auf einzelne Objekte oder Benutzer beschränkt. Zum Beispiel das Anlegen neuer Anwendungen oder das Verwalten von Benutzern.

Assigned Object Permissions: Hier können alle der Rolle zugewiesenen Berechtigungen für die entsprechenden Objekte eingesehen und bearbeitet werden. In Authentik zählen dazu beispielsweise Gruppen, Rollen oder Flows. Dadurch lässt sich nachvollziehen, welche Objekte von Personen mit dieser Rolle bearbeitet werden können und in welchem Umfang.

User Object Permissions: In diesem Menüpunkt wird festgelegt, auf welche Benutzer die zuvor zugewiesenen Berechtigungen angewendet werden dürfen.

Role Object Permissions: Im Menüpunkt „Role Object Permissions“ wird festgelegt, auf welche Rollen, die Objekt bezogenen Berechtigungen angewendet werden dürfen.

4.2.3.2 Gruppen

Gruppen dienen im Wesentlichen dazu, mehrere Authentik Benutzer in eine gemeinsame Gruppe zusammenzufassen. Diesen Gruppen können dann mithilfe von Rollen die jeweiligen Berechtigungen zugewiesen werden. Gruppen selbst können keine spezifischen Berechtigungen direkt zugewiesen

werden. Stattdessen besteht die Möglichkeit, eine Gruppe als Administratorgruppe zu definieren, wodurch allen Mitgliedern dieser Gruppe automatisch Administratorrechte zugewiesen werden.

4.2.4 Authentifizierung

Die Authentifizierung in Authentik ist in zwei Hauptbereiche unterteilt: **Anwendungen** und **Anbieter**. Anwendungen beziehen sich auf die Dienste oder Ressourcen, auf die Benutzer zugreifen möchten, während Anbieter die Systeme oder Methoden repräsentieren, mit denen die Benutzer ihre Identität verifizieren können.

4.2.4.1 Anwendungen

Als Anwendungen bezeichnet man Dienste und Ressourcen, auf die, die Benutzer zugreifen können und die mithilfe von Authentik authentifiziert werden. Diese Anwendungen werden auf dem Dashboard als Kacheln angezeigt, wie auf Abbildung 15 zu sehen ist. Man kann mithilfe von Rollen und Gruppen präzise bestimmen, welche Benutzer auf bestimmte Anwendungen zugreifen oder nicht zugreifen können.

4.2.4.2 Anbieter

Unter einem Anbieter versteht man die Authentifizierungsmethode, die von einer Anwendung verwendet wird, um den Benutzer bei der jeweiligen Anwendung zu authentifizieren. Im Regelfall stehen die Anbieter in einer 1 zu 1 Beziehung zu einer Anwendung. Das heißt, jede Anwendung muss einen Anbieter haben und jeder Anbieter wird einer Anwendung zugeordnet. Für bestimmte Authentifizierungsanbieter wie *OAuth2* oder *OpenID* muss dem Anbieter ein zusätzlicher Ablauf zugewiesen werden. Abläufe werden genauer im Kapitel *Abläufe* erklärt. In dieser Arbeit wurde für die Authentifizierung der Dienste der *OpenID* Standard verwendet.

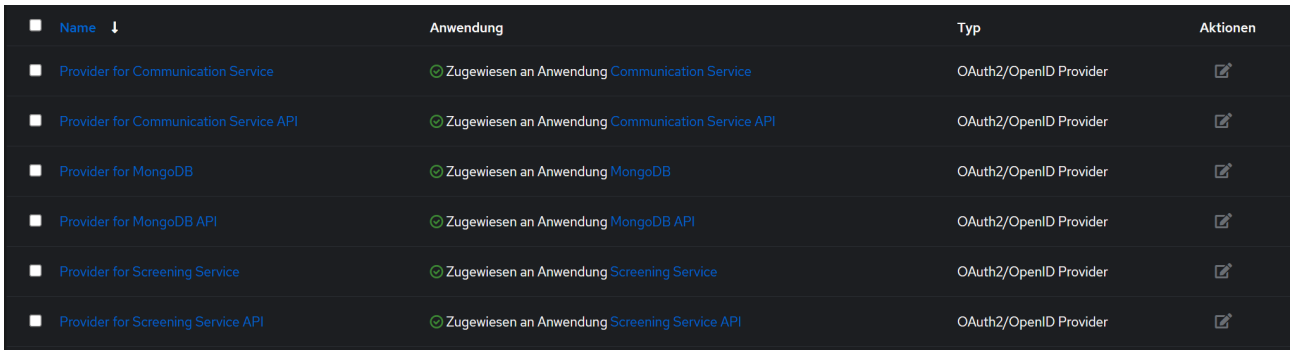


Abbildung 19: Authentik Anbieterverwaltung

4.2.4.3 Abläufe

Abläufe in Authentik sind strukturierte Methoden, die den Prozess der Authentifizierung durch eine Abfolge von Authentifizierungsebenen steuern. Jede Authentifizierungsebene stellt dabei einen einzelnen Schritt innerhalb der Benutzeridentifikation dar. Diese Ebenen ermöglichen eine flexible und anpassbare Authentifizierungsstrategie, die je nach Sicherheitsanforderungen variieren kann.

Beispiele für Authentifizierungsebenen:

- **Identifikationsebene:** In dieser ersten Stufe gibt der Benutzer entweder seinen Benutzernamen oder seine E-Mail-Adresse ein, um sich zu identifizieren. Dies dient als Grundlage für die weiteren Authentifizierungsschritte.
- **Passwortebene:** Falls erforderlich, wird der Benutzer aufgefordert, sein Passwort einzugeben. Dieses wird anschließend mit den gespeicherten Zugangsdaten abgeglichen, um die Identität zu verifizieren.
- **Login-Ebene:** Sobald alle vorherigen Überprüfungen erfolgreich abgeschlossen wurden, wird der Benutzer in die aktuelle Authentik-Session aufgenommen. Dadurch erhält er Zugriff auf die autorisierten Anwendungen und Ressourcen.

Durch diese modulare Struktur kann Authentik je nach Sicherheitsanforderungen zusätzliche Ebenen wie eine Multi-Faktor-Authentifizierung (MFA) oder andere benutzerdefinierte Prüfungen integrieren, um den Schutz weiter zu erhöhen.

In Abbildung 20 ist eine Skizze des Standart Ablaufes (*default-authentication-flow*) in Authentik zu sehen. Dieser Ablauf wurde in allen Anwendungen dieser Arbeit verwendet. Der Prozess beginnt mit einer Identifikationsebene, bei der, der Benutzer E-Mail oder Benutzername eingeben muss. Darauf folgt eine Überprüfung, ob eine Passworteingabe notwendig ist (z.B. aufgrund von vorherigen Anmeldungen oder bestimmten Nutzergruppen, die kein Passwort haben.) Wird ein Passwort benötigt, muss der Benutzer dieses eingeben und wenn die Passwortüberprüfung korrekt ist, wird, falls benötigt, eine MFA geprüft. Danach wird der Benutzer in Authentik eingeloggt.

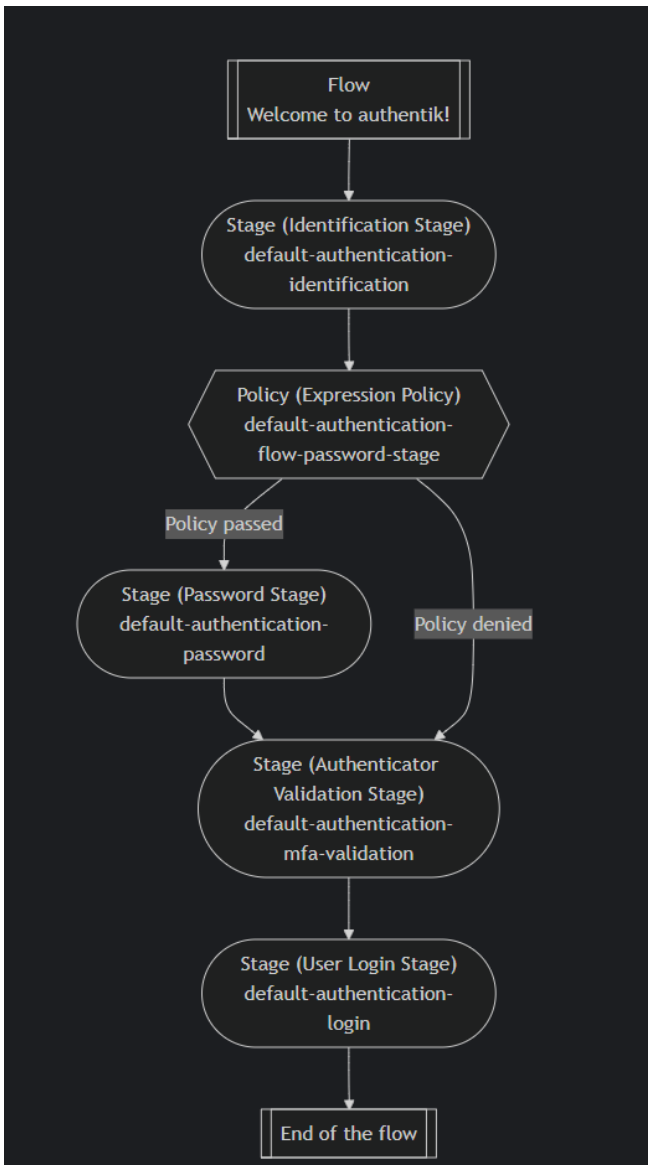


Abbildung 20: Authentik Standard Ablauf

4.3 Module

4.3.1 Einbindung von Modulen als Submodul

Modulstruktur: Das Modul ist als Ordner mit dem Namen `<modulename>_module` aufgebaut. Dieser wird im Projekt-Root eingebunden, wie unten dargestellt [65]:

```
/app
|-- main.py
|-- <modulename>_module/
|   |-- __init__.py
|   |-- <module>.py
```

```
|   |-- config.py
|   |-- requirements.txt
|   |-- README.md
|-- routers/
    |-- protected_routes.py
```

Python-Abhängigkeiten: Für das Modul werden bestimmte Bibliotheken benötigt. Ein Beispiel für die Datei `requirements.txt` könnte wie folgt aussehen [66]:

```
httpx
python-jose[cryptography]
fastapi<
```

1. `.gitmodules` Datei erstellen Die Datei `.gitmodules` wird automatisch erstellt, wenn das Submodul hinzugefügt wird.

2. Submodul hinzufügen (per SSH)

```
git submodule add git@github.com:username/<modulename>-module.git <modulename>_module/
```

3. Submodule initialisieren

```
git submodule update --init --recursive
```

4. SSH-URL auf HTTPS ändern

```
[submodule "<modulename>_module"]
    path = <modulename>_module
    url = https://github.com/username/<modulename>-module.git
```

5. Änderungen committen

```
git add .gitmodules <modulename>_module/
git commit -m "Add <modulename> module as submodule and switch to HTTPS"
```

6. Dockerfile anpassen COPY-Anweisung:

```
COPY ./<modulename>_module /srv/<modulename>_module
```

ENV-Variablen setzen: Ausführung in den einzelnen Modulen, da jedes andere benötigt.

Abhängigkeiten installieren:

```
RUN pip install -r /srv/<modulename>_module/requirements.txt
```

7. `.onedev-buildspec.yml` anpassen Im OneDev-Browsereditor:

- **Retrieve Submodules**: aktivieren
- **Clone Credential**: `HTTP(S)` auswählen
- **Access Token Secret**: `user_token` auswählen

Speichere die Datei anschließend.

4.3.2 Authentication-Modul

Dieses Modul ermöglicht die Authentifizierung und Validierung von JWT-Tokens über OpenID Connect. Es greift dazu auf den JWKS-Endpoint eines Authentifizierungsservers zurück, um öffentliche Schlüssel zur Verifizierung zu laden [1, 67].

4.3.2.1 Übersicht

Das Modul besteht im Wesentlichen aus zwei zentralen Funktionen:

- `get_public_key()`: Lädt und validiert die Public Keys vom JWKS-Endpoint .
- `verify_token()`: Überprüft die Gültigkeit eines übermittelten JWTs [68].

Zusätzlich ist ein robustes Error Handling implementiert, das auf spezifische Fehlerquellen eingeht.

4.3.2.2 Verwendete Pakete

Die Verifizierung basiert auf dem Paket `python-jose`, welches JWTs dekodieren und anhand von JWKS validieren kann.
Notwendige Imports:

```
from jose import jwt, JWTError, jwk
from jose.exceptions import JWKError
from fastapi import HTTPException, Depends
from fastapi.security import OAuth2PasswordBearer
import httpx, asyncio, logging, os

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")
```

4.3.2.3 Funktion `get_public_key()`

Diese asynchrone Funktion ruft die Public Keys vom JWKS-Endpoint ab, welcher über die Umgebungsvariable `jwks_url` gesetzt ist. Die Anfrage wird bis zu drei Mal wiederholt, bevor eine Fehlermeldung geworfen wird. Die Implementierung verwendet das `httpx`-Modul für asynchrone HTTP-Requests [69].

```
async def get_public_key():
    async with httpx.AsyncClient(verify=False) as client:
        for attempt in range(3):
            try:
                response = await client.get(jwks_url)
                response.raise_for_status()
                response_json = response.json()

                if "keys" in response_json:
                    return response_json
                else:
                    logging.error(f"Invalid response format on attempt
                    ↪ {attempt + 1}: {response_json}")

            except httpx.HTTPStatusError as e:
                logging.error(f"HTTPError on attempt {attempt + 1}: {e}")
            except httpx.RequestError as e:
                logging.error(f"RequestError on attempt {attempt + 1}: {e}")
            except httpx.JSONDecodeError as e:
```

```
            logging.error(f"JSONDecodeError on attempt {attempt + 1}:
            ↪ {e}")
        except httpx.Error as e:
            logging.error(f"Unexpected error on attempt {attempt + 1}:
            ↪ {e}")

        await asyncio.sleep(2)

raise HTTPException(status_code=503, detail="Unable to fetch public keys
↪ from JWKS endpoint")
```

Fehlerbehandlung:

- `HTTPStatusError`: Bei fehlerhaftem HTTP-Statuscode
- `RequestError`: Netzwerkprobleme oder DNS-Fehler
- `JSONDecodeError`: Antwort war kein valides JSON
- `httpx.Error`: Alle weiteren httpx-bezogenen Fehler

4.3.2.4 Funktion `verify_token()`

Diese Funktion prüft ein JWT anhand der zuvor geladenen Public Keys. Sie extrahiert den Key-Identifier (`kid`) aus dem Header und nutzt diesen zur Auswahl des richtigen öffentlichen Schlüssels.

```
async def verify_token(token: str = Depends(oauth2_scheme)):
    try:
        public_key = await get_public_key()
        headers = jwt.get_unverified_header(token)
        key_id = headers.get("kid")

        key = next((key for key in public_key["keys"] if key["kid"] ==
        ↪ key_id), None)
        if key is None:
            raise HTTPException(status_code=401, detail="Invalid credentials:
            ↪ Key ID not found in JWKS")
```

```
        payload = jwt.decode(token, key, algorithms=["RS256"],
        ↪  audience=clientID)
        user = payload.get("sub")
        if user is None:
            raise HTTPException(status_code=401, detail="Invalid credentials:
            ↪  User not found in token")

        return user

    except JWTError as e:
        raise HTTPException(status_code=401, detail="Invalid credentials: JWT
        ↪  error")
    except JWKError as e:
        raise HTTPException(status_code=401, detail="Invalid credentials: JWK
        ↪  processing error")
    except Exception as e:
        logging.error(f"Unexpected error during token verification: {e}")
        raise HTTPException(status_code=500, detail="Internal server error")
```

Fehlerbehandlung:

- `JWTError`: Token ungültig oder nicht verifizierbar
- `JWKError`: Fehler bei der Verarbeitung der JWKS
- `HTTPException (401)`: Key-ID nicht vorhanden oder Benutzer fehlt
- `HTTPException (500)`: Unerwarteter interner Fehler

4.3.2.5 Konfiguration und Setup

Die Umgebungsvariablen werden in der `Dockerfile` bzw. beim Container-Start gesetzt [70]:

```
ENV jwks_url=https://auth.example.com/application/o/jwks/
ENV oidc_aud=my-fastapi-backend
ENV oidc_url=https://auth.example.com/application/o/.well-known/openid-configuration
ENV token_url=https://auth.example.com/application/o/token/
ENV auth_url=https://auth.example.com/application/o/authorize/
```

4.3.2.6 Verwendung im Modul

```
jwks_url = os.environ['jwks_url']
clientID = os.environ['oidc_aud']
oidc_url = os.environ['oidc_url']
token_url = os.environ['token_url']
auth_url = os.environ['auth_url']
```

4.3.2.7 Integration in FastAPI-Projekt

Das Authentication-Modul kann in FastAPI-Projekten verwendet werden, um geschützte Routen per Token-Authentifizierung abzusichern [71]:

```
from authentication_module.token import verify_token
from fastapi import APIRouter, Depends

router = APIRouter()

@router.get("/protected-route")
async def protected_route(user: str = Depends(verify_token)):
    return {"message": f"Hello {user}, you are authenticated!"}
```

4.3.3 Secret-Communication-Module

Das Secret-Communication-Module stellt Funktionen zur sicheren Verwaltung und zum Abruf sensibler Daten aus Kubernetes Secret bereit. Dieses Modul wird insbesondere vom **Database-Module** wie auch vom **Communication-Service** genutzt, um MongoDB-URLs bzw Communication-Service-URLs sicher zu speichern und zu verwalten.

Das Modul nutzt offizielle Kubernetes-Mechanismen zur sicheren Secret-Verwaltung über die Kubernetes-API [72]. Die Verbindung erfolgt mithilfe des Python Kubernetes Clients, der Zugriff über Bearer-Tokens erlaubt [73]. Die Secrets selbst werden im Kubernetes-Cluster Base64-kodiert gespeichert [74].

4.3.3.1 Verbindung mit Kubernetes Secrets aufbauen

Um eine sichere Verbindung zu den Kubernetes-Secrets herzustellen, nutzt das Secret-Communication-Module die offizielle `kubernetes`-Python-Clientbibliothek. Die Authentifizierung erfolgt über einen Bearer-Token sowie die API-URL des Kubernetes-Clusters. Diese werden dynamisch über Umgebungsvariablen eingelesen, wie in der `secretCommModule_config.py` Datei definiert:

```python
# secretCommModule_config.py

import os

kubernetes_con = {
    "host": os.environ['KUBERNETES_CLUSTER_HOST'],
    "bearer_token": os.environ['KUBERNETES_BEARER_TOKEN']
}
```

Im Initialisierungsteil des Moduls wird eine individuelle `ApiClient`-Instanz mit diesen Werten erstellt. Anschließend kann über das `CoreV1Api`-Interface auf Secrets zugegriffen werden:

```python
from kubernetes import client

aToken = kubernetes_con['bearer_token']
aHost = kubernetes_con['host']

aConfiguration = client.Configuration()
aConfiguration.host = aHost
aConfiguration.verify_ssl = False  # Hinweis: ggf. konfigurierbar machen
aConfiguration.api_key = {"authorization": "Bearer " + aToken}

aApiClient = client.ApiClient(aConfiguration)
core_v1 = client.CoreV1Api(aApiClient)
```

Die Verbindung erlaubt das Lesen und Aktualisieren von Secrets über die Methoden `read_namespaced_secret` und `patch_namespaced_secret`. Die Secret-Inhalte werden Base64-dekodiert, als JSON geparst und ggf. nach der Bearbeitung erneut Base64-kodiert zurückgeschrieben.

Sicherheitsaspekte

- Der Zugriff auf Kubernetes-Secrets erfolgt ausschließlich über einen gültigen Bearer-Token, der zur Laufzeit über eine Umgebungsvariable bereitgestellt wird.
- Nur Clients mit entsprechender Kubernetes-RBAC-Berechtigung (z. B. `get`, `patch` auf Secrets) können auf die sensiblen Daten zugreifen.

Fehlerbehandlung Die strukturierte Fehlerbehandlung orientiert sich an gängigen HTTP-Statuscodes, um eine einfache Verarbeitung im Client oder Frontend zu ermöglichen [75].

- Alle Funktionen geben strukturierte Fehlerobjekte mit HTTP-ähnlichen Statuscodes zurück (403, 404, 400, 500) sowie klar definierte Fehlermeldungen.
- Fehler beim Parsen der Secret-Daten werden mit `json_decode_error` und dem Code 400 behandelt.
- Dadurch ist eine robuste Weiterverarbeitung in aufrufenden Services möglich.

Lese- vs. Schreiboperationen

- Das Modul trennt strikt zwischen Lese- (`retrieve_*`) und Schreiboperationen (`set_`, `put_`, `delete_`).
- Leseoperationen verändern den Clusterzustand nicht, während Schreiboperationen gezielt Secrets manipulieren und daher mit Bedacht eingesetzt werden sollten.

4.3.4 Funktionen zur Secret-Verwaltung

Abrufen aller Secret-Werte

```
def retrieve_secret_service_urls(secret_item,namespace, secret_name):
    try:
        secret = core_v1.read_namespaced_secret(name=secret_name,
        ↪  namespace=namespace)
        cloud_ep_value = b64decode(secret.data[secret_item]).decode('utf-8')
        cloud_ep_data = json.loads(cloud_ep_value)
        return {"services": cloud_ep_data}
    except client.exceptions.ApiException as e:
        if e.status == 403:
            return {"status_code": 403, "error": "resource_permission_error"}
        elif e.status == 404:
```

```
            return {"status_code": 404, "error": "secret_not_found_error"}
        else:
            return {"status_code": e.status, "error": str(e)}
    except json.JSONDecodeError:
        return {"status_code": 400, "error": "json_decode_error"}
    except Exception as e:
        return {"status_code": 500, "error": "unexpected_error"}
```

Diese Funktion ermöglicht das Abrufen aller in Kubernetes gespeicherten Secret-Service-URLs für einen bestimmten Namespace. Dafür müssen der Namespace, der Name des Secret-Files sowie die Spalte, in der die URLs gespeichert sind, als Parameter angegeben werden. Die Funktion liest die entsprechenden Werte aus den Kubernetes-Secrets aus und gibt sie zurück. Falls ein Fehler auftritt, beispielsweise wenn das Secret nicht existiert oder nicht korrekt ausgelesen werden kann, wird eine Fehlermeldung mit einem entsprechenden HTTP-Statuscode zurückgegeben.

Abrufen einer einzelnen Secret-URL

```
def retrieve_secret_service_url(secret_item, namespace, secret_name, name):
    try:
        secret = core_v1.read_namespaced_secret(name=secret_name,
        ↪  namespace=namespace)
        cloud_ep_value = b64decode(secret.data[secret_item]).decode('utf-8')
        cloud_ep_data = json.loads(cloud_ep_value)
        service = next((item for item in cloud_ep_data if item['name'] ==
        ↪  name), None)
        return {"service": service} if service else {"error":
        ↪  "service_not_found"}
    except client.exceptions.ApiException as e:
        if e.status == 403:
            return {"status_code": 403, "error": "resource_permission_error"}
        elif e.status == 404:
            return {"status_code": 404, "error": "secret_not_found_error"}
        else:
            return {"status_code": e.status, "error": str(e)}
    except json.JSONDecodeError:
        return {"status_code": 400, "error": "json_decode_error"}
```

```
    except Exception as e:
        return {"status_code": 500, "error": "unexpected_error"}
```

Diese Funktion ermöglicht das Abrufen einer einzelnen Secret-Service-URL anhand ihres Namens. Zusätzlich zu den Parametern des vorherigen Endpoints muss der Parameter name angegeben werden, um eine spezifische URL innerhalb des Secrets zu identifizieren. Die Funktion durchsucht die Secret-Daten nach dem angegebenen Namen und gibt die zugehörige URL zurück. Falls die URL nicht gefunden wird oder ein anderer Fehler auftritt, wird eine HTTP-Exception mit einer entsprechenden Fehlermeldung ausgelöst.

Speichern einer neuen Secret-URL

```
def set_secret_service_url(secret_item, namespace, secret_name, name,
↪  new_api_url, active):
    try:
        secret = core_v1.read_namespaced_secret(name=secret_name,
        ↪  namespace=namespace)
        current_value = b64decode(secret.data.get(secret_item,
        ↪  '')).decode('utf-8')
        comm_services = json.loads(current_value) if current_value else []
        service = next((item for item in comm_services if item['name'] ==
        ↪  name), None)
        if service:
            service['url'] = new_api_url
            service['active'] = active
        else:
            comm_services.append({"name": name, "url": new_api_url, "active":
            ↪  active})
        updated_value = json.dumps(comm_services)
        secret.data[secret_item] =
        ↪  b64encode(updated_value.encode('utf-8')).decode('utf-8')
        core_v1.patch_namespaced_secret(name=secret_name,
        ↪  namespace=namespace, body=secret)
        return {"success": "Secret service URL updated successfully."}
    except client.exceptions.ApiException as e:
        if e.status == 403:
```

```
            return {"status_code": 403, "error": "resource_permission_error"}
        elif e.status == 404:
            return {"status_code": 404, "error": "secret_not_found_error"}
        else:
            return {"status_code": e.status, "error": str(e)}
    except json.JSONDecodeError:
        return {"status_code": 400, "error": "json_decode_error"}
    except Exception as e:
        return {"status_code": 500, "error": "unexpected_error"}
```

Diese Funktion dient dazu, eine neue Secret-Service-URL in Kubernetes zu speichern. Dafür müssen der Namespace, der Name des Secret-Files, die Spalte sowie der Name der neuen URL angegeben werden. Zusätzlich wird die tatsächliche URL und ein active-Parameter übergeben, der bestimmt, ob die URL aktiv ist. Die Funktion speichert diese Daten im Kubernetes-Secret-Management-System. Falls die Speicherung fehlschlägt, wird eine HTTP-Exception mit einer detaillierten Fehlermeldung zurückgegeben.

Löschen einer Secret-URL

```
def delete_secret_service_url(secret_item, namespace, secret_name, name):
    try:
        secret = core_v1.read_namespaced_secret(name=secret_name,
        ↪  namespace=namespace)
        current_value = b64decode(secret.data.get(secret_item,
        ↪  '')).decode('utf-8')
        comm_services = json.loads(current_value) if current_value else []
        service = next((item for item in comm_services if item['name'] ==
        ↪  name), None)
        if service:
            comm_services.remove(service)
            updated_value = json.dumps(comm_services)
            secret.data[secret_item] =
            ↪  b64encode(updated_value.encode('utf-8')).decode('utf-8')
            core_v1.patch_namespaced_secret(name=secret_name,
            ↪  namespace=namespace, body=secret)
            return {"success": "Service deleted successfully."}
```

```
        else:
            return {"status_code": 404, "error": "service_not_found_error"}
    except client.exceptions.ApiException as e:
        if e.status == 403:
            return {"status_code": 403, "error": "resource_permission_error"}
        elif e.status == 404:
            return {"status_code": 404, "error": "secret_not_found_error"}
        else:
            return {"status_code": e.status, "error": str(e)}
    except json.JSONDecodeError:
        return {"status_code": 400, "error": "json_decode_error"}
    except Exception as e:
        return {"status_code": 500, "error": "unexpected_error"}
```

Diese Funktion ermöglicht das Entfernen einer bestimmten Secret-Service-URL aus Kubernetes. Dazu werden der Namespace, der Name des Secret-Files, die Spalte und der Name der zu löschenden URL als Parameter benötigt. Die Funktion sucht den Eintrag in den Secrets und entfernt ihn. Falls das Secret oder die URL nicht gefunden wird oder ein Fehler bei der Löschung auftritt, wird eine HTTP-Exception mit einer entsprechenden Fehlermeldung geworfen.

Aktualisieren des Aktivitätsstatus einer URL

```
def put_secret_service_url(secret_item, namespace, secret_name, name, active,
↪   new_api_url=None):
    try:
        secret = core_v1.read_namespaced_secret(name=secret_name,
        ↪   namespace=namespace)
        current_value = b64decode(secret.data.get(secret_item,
        ↪   '')).decode('utf-8')
        comm_services = json.loads(current_value) if current_value else []
        service = next((item for item in comm_services if item['name'] ==
        ↪   name), None)
        if service:
            if new_api_url is not None:
                service['url'] = new_api_url
            service['active'] = active
```

```
            updated_value = json.dumps(comm_services)
            secret.data[secret_item] =
            ↪  b64encode(updated_value.encode('utf-8')).decode('utf-8')
            core_v1.patch_namespaced_secret(name=secret_name,
            ↪  namespace=namespace, body=secret)
            return {"success": "Service availability updated successfully."}
        else:
            return {"status_code": 404, "error": "service_not_found_error"}
    except client.exceptions.ApiException as e:
        if e.status == 403:
            return {"status_code": 403, "error": "resource_permission_error"}
        elif e.status == 404:
            return {"status_code": 404, "error": "secret_not_found_error"}
        else:
            return {"status_code": e.status, "error": str(e)}
    except json.JSONDecodeError:
        return {"status_code": 400, "error": "json_decode_error"}
    except Exception as e:
        return {"status_code": 500, "error": "unexpected_error"}
```

Diese Funktion erlaubt es, den active-Status einer Secret-Service-URL zu ändern. Der Namespace, der Name des Secret-Files, die Spalte und der Name der URL müssen als Parameter angegeben werden. Zusätzlich wird der neue active-Wert übergeben. Falls der active-Wert auf false gesetzt wird, wird die URL nicht mehr für Datenbankoperationen verwendet. Falls ein Fehler bei der Aktualisierung auftritt, wird eine HTTP-Exception mit einer detaillierten Fehlermeldung zurückgegeben.

4.3.4.1 Konfiguration und Setup

Um das Modul zu importieren muss man die Schritte wie in 4.3.1 ausführen. Die benötigten Environment Variables sind:

```
ENV KUBERNETES_CLUSTER_HOST=
↪  https://rancher.res.fabagl.fabasoft.com/k8s/clusters/local
ENV KUBERNETES_BEARER_TOKEN=
↪  token-pxndj:lzc2zdqvqwdmsz9vxtcwzzcgwhnhzrcsbksctxrjngvqh2z6xktnnv
```

Der Zugriffstoken wird in Rancher automatisch rotiert oder nach Ablauf ungültig und muss regelmäßig erneuert werden [76].

Import und Einbindung des Moduls im Code Das Secret-Communication-Modul ist als eigenständiges Modul organisiert. Es ermöglicht den sicheren Zugriff und die Verwaltung von API-Endpunkten, die in Kubernetes-Secrets gespeichert sind. In FastAPI-Projekten kann es verwendet werden, um dynamisch URLs aus Kubernetes-Secrets auszulesen, hinzuzufügen, zu ändern oder zu löschen.

Die zentralen Funktionen wie `retrieve_secret_service_url()`, `retrieve_secret_service_urls()`, `set_secret_service_url()`, `delete_secret_service_url()` und `put_secret_service_url()` können direkt in den API-Routen verwendet werden.

In Communaction-Service:

```
from libraries.secretCommModule.secretCommunication import (
    retrieve_secret_service_url,
    retrieve_secret_service_urls,
    set_secret_service_url,
    delete_secret_service_url,
    put_secret_service_url
)
```

Diese Funktionen ermöglichen es, dynamisch Dienste zu verwalten, ohne harte Kodierung von URLs. Die Informationen werden sicher über Kubernetes-Secrets abgelegt und ausgelesen.

4.3.5 Database-Module

Das Database-Module stellt einen Router mit API-Endpoints zur Verwaltung von MongoDB-URLs bereit und ermöglicht das Zugreifen auf mehrere MongoDB-Instanzen. Zur Implementierung der API-Endpoints wird FastAPI genutzt, und PyMongo für den Zugriff auf die Datenbanken.

4.3.5.1 Konfiguration und Setup

Um das Modul zu importieren müssen die Schritte aus Abschnitt 4.3.1 befolgt werden. Die folgende Umgebungsvariable wird benötigt, damit das Database-Module wie gewünscht funktioniert:

```
ENV MONGO_EP = urls
```

4.3.5.2 API-Endpoints

Die API-Routes werden im Modul `database-routes.py` definiert und stellen Funktionen zum Hinzufügen, Löschen und Abrufen von MongoDB-URLs bereit. Alle Endpoints werden mithilfe der Funktion `verify_token` des Authentication-Modules abgesichert.

Der Router wurde folgendermaßen implementiert:

```
router = APIRouter()
```

Diese Zeile definiert den API-Router, welcher die Endpoints enthält. Ein Endpoint des Database-Modules ist im folgenden Code-Snippet ersichtlich:

```
@router.get("/mongodb-urls", description="Get the url of the MongoDB
↪  endpoints", tags=["Database"])
async def get_mongodb_urls(namespace:str,secret_name:str, secret_column:str,
↪  user: str = Depends(verify_token)):
    result = retrieve_secret_service_urls(secret_column,namespace,
    ↪  secret_name)
    if 'error' in result:
        raise HTTPException(status_code=result['status_code'],
        ↪  detail=result['error'])
    return result
```

Der GET-Endpoint `/mongodb_urls` ermöglicht das Abrufen aller MongoDB-URLs, die in den Kubernetes-Secrets gespeichert werden. Der Abruf aus den Kubernetes-Secrets erfolgt durch die Nutzung der `retrieve_secret_service_urls`-Funktion aus dem Secret-Communication-Module. Dazu ist die Angabe des Kubernetes-Namespaces, dem Namen des Secret-Files sowie des Secret-Columns als Parameter erforderlich. Zusätzlich wird der Parameter `user` genutzt, um den Endpoint abzusichern. Falls die Funktion einen Fehler zurückgibt, wird eine HTTP-Exception mit dem entsprechenden Status Code ausgelöst. Die nachfolgenden Router-Endpoints nutzen die gleichen Methoden zur Absicherung der Endpoints und zur Fehlerbehandlung.

Der GET-Endpoint `/mongodb-url` ermöglicht das Abrufen einer einzelnen MongoDB-URL. Dazu wird die Funktion `retrieve_secret_service_url` aus dem Secret-Communication-Module heran-

gezogen, welche zusätzlich zu den Parametern aus dem vorher erwähnten Endpoint die Angabe des Parameters `name`, welcher eine URL identifiziert, erfordert.

Der POST-Endpoint `/mongodb-url` ermöglicht das Hinzufügen von neuen MongoDB-URLs. Um eine neue URL hinzuzufügen, sind neben `secret_column`, `namespace`, `secret_name` und `name` zwei neue Parameter erforderlich: `new_api_url` und `active`. `new_api_url` ist die URL der MongoDB Instanz, und `active` steht für die Verfügbarkeit dieser Instanz. Diese Parameter werden dann der Funktion `set_secret_service_url` aus dem Secret-Communication-Module übergeben, um die URL hinzuzufügen. Verbesserungspotenzial besteht bei diesem Endpoint bei der Validierung: Es wird nicht überprüft, ob die MongoDB-URL dem korrekten Format entspricht. Eine korrekte Überprüfung der MongoDB-URL würde potenzielle Sicherheitslücken schließen, Fehlerquellen vermeiden und damit den Endpoint robuster gestalten.

Um eine ausgewählte MongoDB-URL aus den Kubernetes-Secret zu löschen, wurde ein DELETE Endpoint, `/mongodb-url`, implementiert. Dieser nutzt die Funktion `delete_secret_service_url` aus dem Secret-Communication-Module, und ihr werden beim Aufruf die Query-Parameter `namespace`, `secret_name`, `secret_column` und `name` übergeben.

Der PUT-Endpoint `/mongodb-url/{name}` wurde implementiert, um die **Verfügbarkeit** einer MongoDB-URL zu verändern. Dazu gibt man je nach Bedarf `true` oder `false` beim Parameter `active` an. Solange `active` im Kubernetes-Secret den Wert `false` hat, werden dann keine Daten in dieser Datenbank gespeichert. Um den Wert im Kubernetes-Secret zu verändern, wird die `put_secret_service_url`-Funktion aus dem Secret-Communication-Module genutzt. Ihr werden die Parameter `secret_column`, `namespace`, `secret_name`, `name` und `active` übergeben.

4.3.5.3 Ausführung auf mehreren MongoDB-Instanzen

Die Funktion zur Ausführung einer Funktion auf mehreren MongoDB-Instanzen ist in `database_utils.py` implementiert.

```
from libraries.databaseModule.config.databaseModule_config import
↪  mongodb_url
mongodb_urls = json.loads(mongodb_url['url'])
```

In diesem Code-Snippet wird `mongodb_urls` als JSON-Objekt in folgendem Format geladen:

```
[{
    "name": "name",
    "url": "url",
```

```
    "active": true
}]
```

```
def get_available_clients(mongodb_urls):
    clients = []
    for mongodb_info in mongodb_urls:
        if mongodb_info['active']:
            clients.append(MongoClient(mongodb_info['url']))
    return clients
```

Die Funktion `get_available_clients` dient dazu, alle zurzeit aktiven MongoClients in einem Array zu erstellen und zurückzugeben. Eine MongoDB-Instanz gilt dann als aktiv, wenn `active` aus dem vorhin erwähnten JSON-Objekt den Wert `true` hat.

```
def perform_on_all_instances(func, *args, **kwargs):
    results = []
    for client in get_available_clients(mongodb_urls):
        sensorDB = client['sensor_db']
        outliersDB = client['outliers']
        humidityTrendsDB = client['humidity_trends']

        result = func(sensorDB, outliersDB, humidityTrendsDB, *args,
        ↪  **kwargs)
        results.append(result)
    return results
```

Diese Funktion wird genutzt, um eine Funktion `func` auf mehreren Datenbankinstanzen auszuführen, und übergibt ihr darüber hinaus die Datenbanken `sensor_db`, `outliers` und `humidity_trends`. Das `results`-Objekt ist ein Array, welches die Rückgabewerte der Funktion `func` auf allen ausgeführten MongoDB-Instanzen speichert. Die Funktion nutzt außerdem die zuvor erläuterte Funktion `get_available_clients`, damit die Funktion nur auf den verfügbaren Instanzen ausgeführt wird.

4.4 Fast-API Anwendungen

Sowohl bestehende als auch neue Services, welche im Zuge des Authentigrade Projekts überarbeitet und implementiert wurden, verwenden Fast-API für die Implementierung einer schnellen und effizienten

API. Dieses Kapitel erklärt die grundlegenden Konfigurationen welche für alle Fast-API-basierten Services im System gleich sind und wie die Endpoints für diese Services grundsätzlich implementiert werden. Die genauen Implementierungen der Funktionalitäten der Endpoints in den Services werden dann in den jeweiligen Kapiteln der Services erklärt.

4.4.1 API Initialisierung

Für das initialisieren einer Fast-API App muss ein Objekt angelegt werden, welches die *FastAPI*-Klasse implementiert. Beim definieren dieses Objekts können zusätzliche Konfigurationen, wie der Name der API und eine Beschreibung, festgelegt werden. Der folgende Code zeigt, anhand des Communication-Service als Beispiel, wie so ein Objekt angelegt wird:

```
app = FastAPI(
    title="Approve-CAQ",
    description="This API is for receiving sensor data and information
    ↪ messages"
)
```

4.4.2 API Konfiguration

Die APIs aller Services werden um eine CORS-Middleware erweitert, damit Frontend-Anwendung auf die API zugreifen können. Hierfür wird die *add_middleware()*-Methode verwendet, um die *CORSMiddleware* Klasse als Middleware hinzuzufügen. Folgender Code zeigt, wie diese Methode verwendet wird:

```
app.add_middleware(
    CORSMiddleware,
    allow_credentials=True,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Dieser Code wird von allen Services im System, welche Fast-API verwenden, implementiert. In diesem Code ist außerdem zu sehen, dass die *add_middleware()*-Methode genutzt wird, um weitere Konfigurationen festzulegen. Unter anderem wird mit *allow_origins=[“*”]*, *allow_methods=[“*”]* und *allow_headers=[“*”]* festgelegt, dass alle URLs auf die API zugreifen dürfen und dass alle

Request-Header und -Typen erlaubt sind. Mit *allow_credentials=True* wird festgelegt, dass es erlaubt ist Anfragen mit Authentifizierungs-Daten an die API zu senden.

4.4.3 Implementierung von Endpoints

Um einen Endpoint für die API zu definieren, muss eine Funktion implementiert werden, welche dann mit einem speziellen Decorator versehen wird. Der Decorator wird folgendermaßen definiert:

```
@app.<Request>("<Pfad>")
```

Wie in diesem Beispiel zu sehen ist, wird zuerst das *app* Objekt angegeben. Danach folgt die HTTP-Funktion auf welche der Endpoint hören soll. Als Nächstes wird in Klammern und Anführungszeichen der Pfad angegeben, unter welchem der Endpoint erreichbar sein soll. Im Pfad kann mit geschwungenen Klammern ein Pfad-Parameter deklariert werden. Das folgende Beispiel ist aus dem Backend der MongoDB-UI und nutzt diese Pfad-Parameter:

```
@app.get("/CollNames/{dbname}")
async def getCollNames(dbname: str, user: str = Depends(verify_token)):
    colls = getColls(dbname)
    response = {"content": colls}
    return response
```

In diesem Beispiel ist zu sehen, wie der Pfad-Parameter *dbname* definiert wird. Um in der Funktion auf den Pfad-Parameter zuzugreifen muss er auch in den Übergabe-Parametern der Funktion angegeben werden (*dbname: str*).

Neben Pfad-Parametern nutzen die Services auch das Übergeben von Objekten durch Request-Bodies. Auf diese Objekte kann direkt zugegriffen werden, indem man einen Übergabeparameter für das erwartete Objekt in der Funktion definiert. Folgendes Beispiel aus dem Communication-Service zeigt wie ein POST-Endpoint ein Objekt aus einer Anfrage entgegen nimmt:

```
@app.post("/sensor-data", description="Receive SensorData from different
↪  resources", tags=["SensorData"])
async def get_sensor_data(data: SensorData, user: str =
↪  Depends(verify_token)):
    perform_on_all_instances(saveMessage, data.description.replace(" ", "_"),
    ↪  data.model_dump(), "sensor")
    return {"message": "Sensor-Data received successfully."}
```

Dieser Endpoint erwartet hier ein Objekt namens *data* von der Klasse *SensorData*. Sollte ein Client eine Anfrage an den Endpoint senden, ohne ein Objekt von der Klasse *SensorData* mitzusenden, wird die Anfrage automatisch abgelehnt.

4.4.4 Einbinden des Authentication-Modules

Alle Services verwenden das Authentication-Module, welches in Sektion 4.3.2 näher erläutert wird. Um dieses Modul zu nutzen, muss die Funktion *verify_token* aus dem Modul importiert werden. Dann muss in allen Endpoints, welche abgesichert werden sollen, ein neuer Übergabewert nach folgender Struktur definiert werden:

```
user: str = Depends(verify_token)
```

Im Code-Beispiel vom Communication-Service weiter oben ist zu sehen, wie dieser Übergabeparameter in einem vollständigen Endpoint verwendet wird. Durch das Angeben dieses Übergabewerts, erwartet der Endpoint einen Access-Token, welcher dann automatisch überprüft wird. Sollte der Access-Token ungültig oder nicht vorhanden sein gibt der Endpoint automatisch einen 401 Fehler-Code zurück.

4.4.5 Service mit API starten

Um die Fast-API Applikation zu hosten, wird Uvicorn verwendet. Mit der *run()*-Methode der Uvicorn Klasse lässt sich eine Fast-API Applikation sofort starten. Folgender Code zeigt, wie die API des Screening-Services gestartet wird:

```
if __name__ == '__main__':
        uvicorn.run('api:app',host="0.0.0.0", port=8000)
```

Mit *host="0.0.0.0"* wird festgelegt, dass die Applikation auf localhost gestartet wird. In diesem Beispiel wird außerdem der Port 8000 als Port für die API ausgewählt.

4.4.6 Swagger-Webseite von Fast-API

Fast-API stellt bei jeder API standardmäßig zusätzlich eine Swagger-UI-Webseite zur Verfügung. Diese Webseite listet alle Endpoints auf und ermöglicht es diese Endpoints per UI zu testen. Diese Webseite befindet sich immer unter dem Pfad *"/docs"* der API. In Abbildung 21 ist beispielsweise die Swagger-UI vom MongoDB-UI Backend zu sehen.

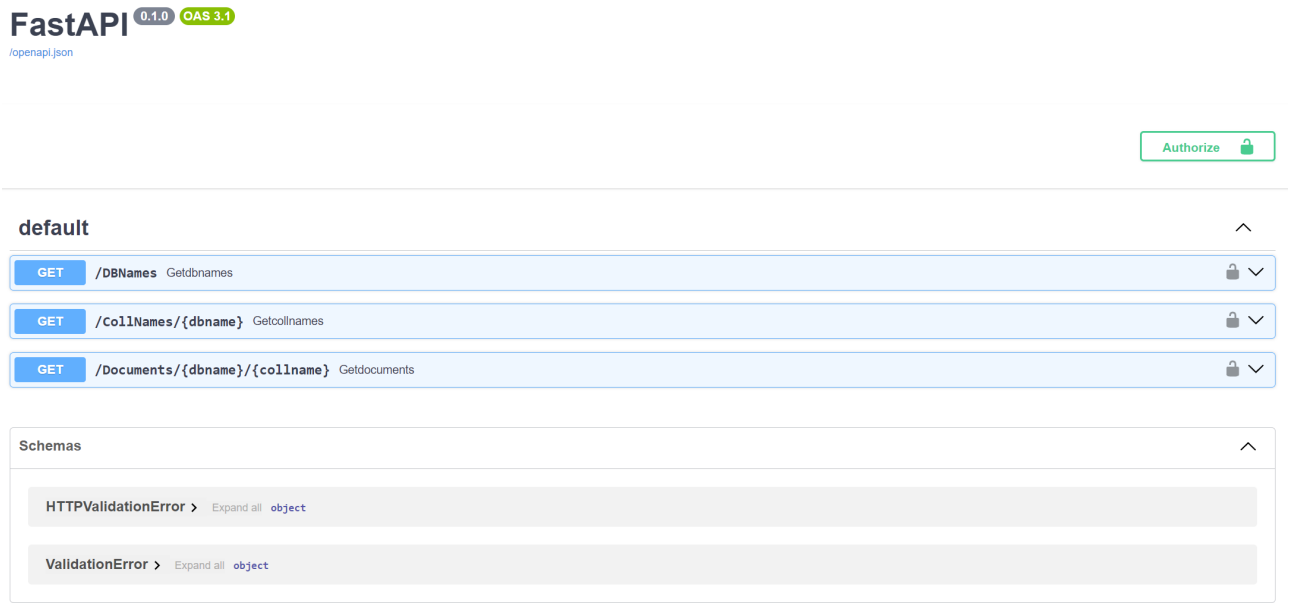


Abbildung 21: Swagger-UI vom MongoDB-UI Backend

4.5 Communication Service

4.5.1 Frontend

Um den Communication Service zu konfigurieren, wird mit dem Frontend-Framework Angular ein Frontend implementiert.

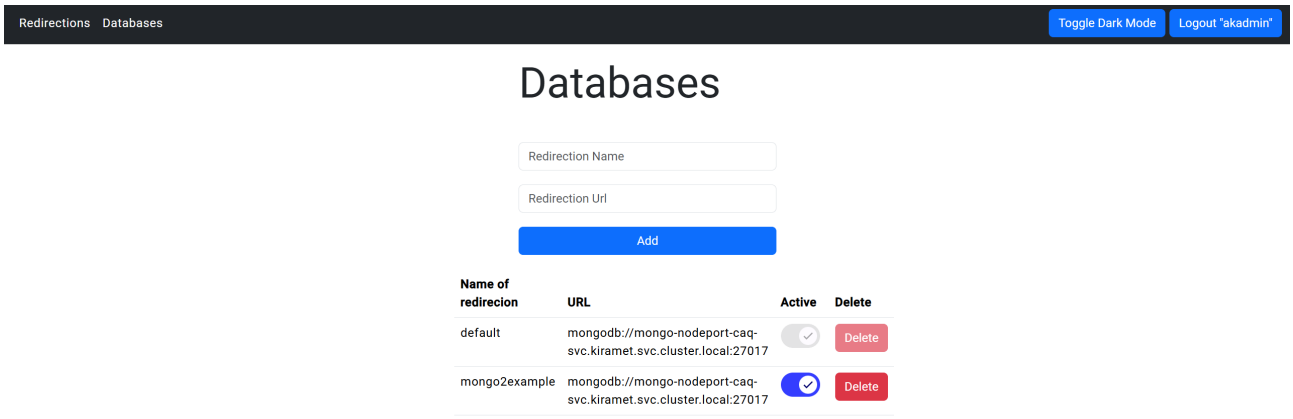


Abbildung 22: CommunicationService: Database Konfiguration

4.5.1.1 Konfiguration

Da das Frontend speziell für den Betrieb in einem Docker-Container konzipiert wird, ist es erforderlich, zentrale Umgebungsvariablen direkt im Dockerfile zu definieren.

Dockerfile

```
ENV API_URL="http://127.0.0.1:5000/"
ENV issuer="https://demo.duendesoftware.com"
ENV clientId="interactive.public"
ENV namespace="kiramet-middleware"
ENV secretName="secrets"
ENV CI=true
```

Um die im Dockerfile angegebenen Variablen in die *env.ts* Datei zu schreiben, wird im Dockerfile ein Shell Skript ausgeführt, das die Variablen in der *env.ts* Datei setzt.

```
RUN chmod +x /usr/local/bin/entrypoint.sh
ENTRYPOINT ["/bin/sh", "-c", "/usr/local/bin/entrypoint.sh"]
```

Shell Skript

```
# Ersetzt die Umgebungsvariablen in der env.ts Datei
# (ersetzt in Wahrheit die Variablen in der main.*.js Datei nach dem
↪ Angular build)
echo "Replacing issuer in environment.ts"
sed -i "s|dummyIssuer1234|$issuer|g" /usr/share/nginx/html/main-*.js
echo "Replacing clientId in environment.ts"
sed -i "s|clientFABASOFT|$clientId|g" /usr/share/nginx/html/main-*.js
echo "Replacing ApiUrl in environment.ts"
sed -i "s|APIURL|$API_URL|g" /usr/share/nginx/html/main-*.js
sed -i "s|namespaceVar|$namespace|g" /usr/share/nginx/html/main-*.js
sed -i "s|secretVar|$secretName|g" /usr/share/nginx/html/main-*.js
```

Die Variablen die in der *env.it* Datei durch das Shell Skript gespeichert wurden, können mit dem Import der Datei im Typescript Code verwendet werden.

Beispiel Verwendung

```
import { environment } from './env';
export const authCodeFlowConfig: AuthConfig = {
  issuer: environment.issuer,
  clientId: environment.clientId,
};
```

4.5.1.2 Authentifizierung

Das Frontend des Communication Service wird mittels des Identity Provider Authentik und dem OpenID Standard authentifiziert. Um OpenID und OIDC in Angular verwenden zu können, muss ein Node Package installiert werden. Hierfür wurde das *angular-oauth2-oidc* Package verwendet.

Das Autorisierungskonzept des Communication Service ist wie folgt: Zunächst erfolgt die Anmeldung des Benutzers in Authentik. Im Anschluss wird der Benutzer auf die Landing-Page des Communication Service weitergeleitet. Dort findet eine Überprüfung statt, ob der Benutzer bereits angemeldet ist. Bei einer positiven Überprüfung erhält der Benutzer Zugriff auf das Frontend des Communication Service. Bei negativer Überprüfung wird der Benutzer wieder auf die Authentik Login Seite geleitet.

Die Überprüfung, ob der Benutzer im Authentik angemeldet ist, wird durch das OIDC Package durchgeführt. Dazu wurde in der *landing-page.component.ts* Datei folgender Code implementiert:

```
import { OAuthService } from 'angular-oauth2-oidc';
// Import der Authentifizierungs-Konfiguration
import { authCodeFlowConfig } from '../auth.config';
import { Router } from '@angular/router';
export class LandingPageComponent {
  // Benutzer ist nicht angemeldet in -> weiterleiten zum Identity Provider
  constructor(private oauthService: OAuthService, private router: Router) {
    this.oauthService.configure(authCodeFlowConfig);
    this.oauthService.loadDiscoveryDocumentAndLogin().then((hasReceivedToken:
    ↪  any) => {
      // Benutzer ist angemeldet -> wird zum Frontend weitergeleitet
      if (hasReceivedToken) this.router.navigate(['/redirection-config']);
    });
  }
}
```

Die Authentifizierungs-Konfiguration befindet sich in einem eigenen Objekt des Typen *AuthConfig*. Dieses Objekt wird vom Package *angular-oatuh2-oidc* bereitgestellt.

auth.config.ts

```
import { AuthConfig } from 'angular-oauth2-oidc';
import { environment } from './env';
export const authCodeFlowConfig: AuthConfig = {
  issuer: environment.issuer,
  redirectUri: `${window.location.origin}/login`,
  clientId: environment.clientId,
  responseType: 'code',
  scope: 'openid profile email offline_access',
  showDebugInformation: true,
  strictDiscoveryDocumentValidation: false
};
```

4.5.1.3 Aufbau

Das Frontend des Communication-Service verfügt über eine Navigationsleiste am oberen Rand der Benutzeroberfläche. Diese ermöglicht das Wechseln zwischen den Seiten: *Redirections* und *Databases*, das Abmelden vom System sowie das Anpassen des Farbmodus.

Der Tab *Redirections* enthält zwei Eingabefelder, die zur Vergabe eines Namens für die Redirection sowie zur Angabe der entsprechenden URL dienen. Darunter befindet sich ein Button, der die Redirection an das Backend übergibt. Bei einer erfolgreichen Speicherung der Redirection wird diese in der darunterliegenden Tabelle angezeigt. In dieser Tabelle befindet sich pro Eintrag ein Toggle Button, mit dem man eine Redirection temporär deaktivieren kann. Mit einem Delete Button kann jede Redirection auch permanent gelöscht werden.

Der Tab *Databases* ist identisch aufgebaut zum *Redirections* Tab. Deshalb wurde ein eigener Angular Component für die Konfigurations-Tabs erstellt, mit der ein Ändern der Konfiguration-Tabs relativ einfach umsetzbar ist. Diese beinhaltet die beiden Eingabefelder und die Tabelle inklusive Buttons.

Konfigurations Component

```
    <div class="input">
    <h1>{{pageName}}</h1>
    <input [(ngModel)]="newDatabase.url" type="text" class="form-control
    ↪  input-field" placeholder="Redirection Url">
    <button (click)="addUrl()" class="btn btn-primary
    ↪  input-field">Add</button>
  </div>
  <div class="items">
    <table class="table">
      <thead>
        <tr>
          <th scope="col">Name of redirecion</th>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let i of databases; let ind = index">
          <td>{{i.name}}</td>
        </tr>
      </tbody>
    </table>
  </div>
```

In den Komponenten *redirection-config-page* und *database-config-page* wurde dann der vorgefertigte Komponent *config-page* folgendermaßen verwendet:

```
    <app-config-page [service]="'comm-service-url'" [baseURL]="baseUrl"
    ↪  [pageName]="'Message Redirection'" [namespace]="namespace"
    ↪  [secretName]="secretName"></app-config-page>
```

Es müssen einige wichtige Parameter der Komponente übergeben werden. Wie zum Beispiel der Name der Konfigurationsseite. In diesem Fall *Message Redirection*. In Abbildung 23 ist zu sehen, dass der Name der Konfigurationsseite *Message Redirection* ist.

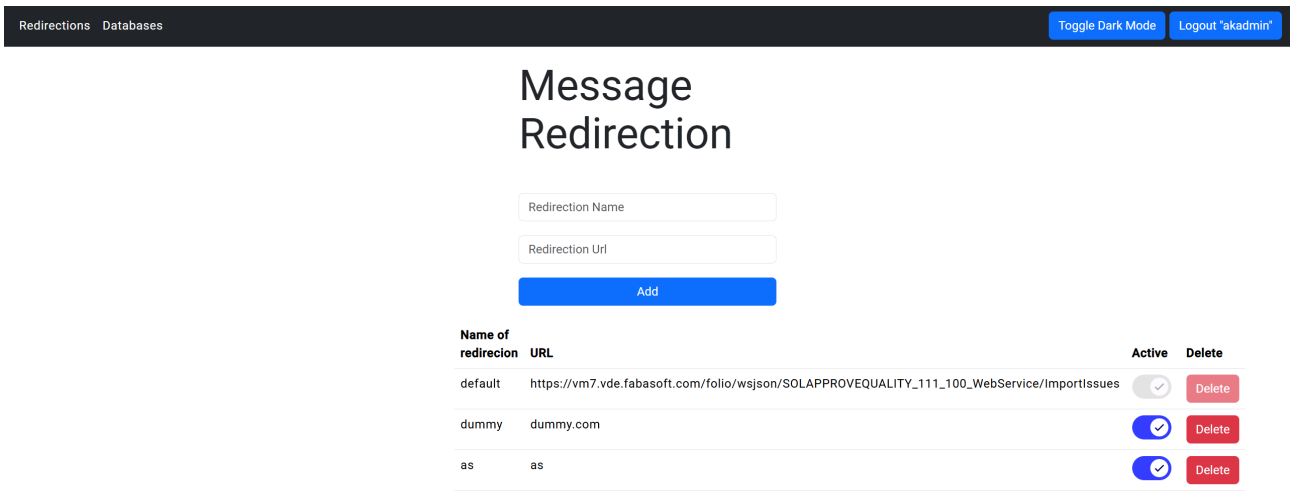


Abbildung 23: Communication-Service Message Redirection

In der Abbildung 23 ist der *Redirections* Tab zu sehen.

4.5.1.4 Design

Um ein einheitliches Design des Frontends sicher zustellen wurde für das Design Bootstrap benutzt. [77]

Button mit Bootstrap

```
<button class="btn btn-primary">Add</button>
```

4.5.2 Backend

Das Communication-Service-Backend stellt Funktionen zur Verarbeitung und Übertragung von Kommunikationsdaten zwischen verschiedenen Ressourcen bereit. Dieses Modul wird hauptsächlich verwendet, um Sensor- und Informationsdaten zu empfangen und weiterzuleiten [78].

Alle API-Endpunkte folgen den Prinzipien REST-konformer Schnittstellen [79] und verwenden standardisierte HTTP-Antwortcodes [75]. Die Authentifizierung erfolgt jeweils über das `verify_token()`-Verfahren was mithilfe von OAuth2-JWT implementiert ist[71].

4.5.2.1 Vorhandene Endpoints

Empfangen von Informationsdaten (`/CAQ`)

```
@app.post("/CAQ", description="Receive Information from different resources",
↪  tags=["CAQ"])
async def get_information_data(data: InformationMessage, namespace: str,
↪  secret_name: str, secret_column: str, , user: str =
↪  Depends(verify_token)):
    endpoints = retrieve_secret_service_urls(secret_column, namespace,
    ↪  secret_name)
    services = endpoints.get("services", [])
    responses = []
    for service in services:
        if service.get("active", False):
            response = requests.post(service["url"], proxies=proxies,
            ↪  json=data.dict(),
            ↪  auth=HTTPBasicAuth(username=endpoint_cloud_url["username"],
            ↪  password=endpoint_cloud_url["password"]))
            if response.status_code != httpx.codes.OK and
            ↪  response.status_code != httpx.codes.INTERNAL_SERVER_ERROR:
                raise HTTPException(status_code=response.status_code,
                ↪  detail="Failed to send data to cloud.")
            responses.append({"message": "Data sent successfully to cloudurl:
            ↪  " + service["url"]})
    return responses
```

In der neuen Version dieses bestehenden Endpoints wurde die ursprüngliche Logik, bei der Daten an einen einzigen, hardcodierten Endpoint gesendet wurden, vollständig ersetzt. Stattdessen werden nun dynamisch mehrere Service-URLs aus Kubernetes-Secrets geladen. Die Kommunikation erfolgt nur mit aktiven Diensten, die in einer Schleife durchlaufen werden. Zusätzlich wurde die Infrastruktur für Authentifizierung vorbereitet.

Empfangen von Sensordaten (`/sensor-data`)

```
@app.post("/sensor-data", description="Receive SensorData from different
↪ resources", tags=["SensorData"])
async def get_sensor_data(data: SensorData, user: str =
↪ Depends(verify_token)):
    perform_on_all_instances(saveMessage, data.description.replace(" ", "_"),
    ↪ data.model_dump(), "sensor")
    return {"message": "Sensor-Data received successfully."}
```

Dieser bereits bestehende Endpoint empfängt Sensordaten und wurde um Authentifizierungsmöglichkeiten ergänzt.

4.5.2.2 Neue Endpoints

Abrufen von Kommunikationsservice-URLs (`/comm-service-urls`)

```
@app.get("/comm-service-urls", description="Get the url of the Communication
↪ service endpoints", tags=["Message"])
async def get_comm_urls(namespace: str, secret_name: str, secret_column: str,
↪ user: str = Depends(verify_token)):
    result = retrieve_secret_service_urls(secret_column, namespace,
    ↪ secret_name)
    if 'error' in result:
        raise HTTPException(status_code=result['status_code'],
        ↪ detail=result['error'])
    return result
```

Gibt alle registrierten Kommunikationsservice-URLs für einen bestimmten Namespace zurück.

Abrufen einer bestimmten Kommunikationsservice-URL (`/comm-service-url`)

```
@app.get("/comm-service-url", description="Get the url of the Communication
↪ service endpoint", tags=["Message"])
async def get_comm_url(secret_name: str, namespace: str, secret_column: str,
↪ name: str, user: str = Depends(verify_token)):
    result = retrieve_secret_service_url(secret_column, namespace,
    ↪ secret_name, name)
```

```
    if 'error' in result:
        raise HTTPException(status_code=result['status_code'],
        ↪  detail=result['error'])
    return result
```

Liefert eine bestimmte URL eines Kommunikationsdienstes, identifiziert über den Namen.

Speichern einer neuen Kommunikationsservice-URL (`/comm-service-url`)

```
@app.post("/comm-service-url", description="Set the url of the Communication
↪  service endpoint", tags=["Message"])
async def set_comm_url(namespace: str, secret_name: str, secret_column: str,
↪  name: str, new_api_url: str, active: bool, user: str =
↪  Depends(verify_token)):
    result = set_secret_service_url(secret_column, namespace, secret_name,
    ↪  name, new_api_url, active)
    if 'error' in result:
        raise HTTPException(status_code=result['status_code'],
        ↪  detail=result['error'])
    return result
```

Speichert eine neue Kommunikationsservice-URL in den Kubernetes-Secrets.

Löschen einer Kommunikationsservice-URL (`DELETE /comm-service-url`)

```
@app.delete("/comm-service-url", description="Delete a url of the
↪  Communication service endpoints", tags=["Message"])
async def delete_comm_url(namespace: str, secret_name: str, secret_column:
↪  str, name: str, user: str = Depends(verify_token)):
    result = delete_secret_service_url(secret_column, namespace, secret_name,
    ↪  name)
    if 'error' in result:
        raise HTTPException(status_code=result['status_code'],
        ↪  detail=result['error'])
    return result
```

Löscht eine bestimmte URL aus den gespeicherten Kommunikationsdiensten.

Aktualisieren des Aktivitätsstatus einer URL (`PUT /comm-service-url/{name}`)

```
@app.put("/comm-service-url/{name}", description="Change a url of the
↪  communication service endpoints", tags=["Message"])
async def update_comm_url(namespace: str, secret_name: str, secret_column:
↪  str, name: str, active: bool, user: str = Depends(verify_token)):
    result = put_secret_service_url(secret_column, namespace, secret_name,
    ↪  name, active)
    if 'error' in result:
        raise HTTPException(status_code=result['status_code'],
        ↪  detail=result['error'])
    return result
```

Aktualisiert den Aktivitätsstatus eines Kommunikationsdienstes.

4.5.2.3 Tests

Es wurden Tests implementiert, um die Gültigkeit der gesendeten Nachrichten an den Communication Service sowie die Sensordaten zu überprüfen. Die `httpx`-Bibliothek wird verwendet, um die Status-Codes zu vergleichen. Sowohl für die Nachrichten als auch die Sensordaten sind Tests mit gültigen und ungültigen Parametern implementiert. Falls ein Test ungültige Parameter oder Datentypen übermittelt, wird der Status-Code 422 (UNPROCESSABLE ENTITY) mit einer Fehlermeldung erwartet, und bei Tests mit gültigem Parameter der Status-Code 200 (OK) mit einer Erfolgsnachricht.

Beispiel: POST-Endpoint `/sensor-data`

```
sensor_data0 = {
    'id': 0,
    'timestamp' : '12.04.2024 10:08:15',
    'description': 'sensor description',
    'value': 5.4,
    'unit': "sensor unit"
}

def test_receive_sensor_data0_success():
response = client.post("/sensor-data", json=sensor_data0)
assert response.status_code == httpx.codes.OK
assert response.json() == {"message": "Sensor-Data received successfully."}
```

In diesem Test wird eine Anfrage mit gültigen Sensordaten an den API-Endpoint gesendet, und eine erfolgreiche Antwort mit dem Status Code 200 (OK) wird erwartet.

Zusätzlich gibt es einen Test für die Authentifizierung und Berechtigung:

- Eine Anfrage mit falschen Zugangsdaten wird mit einem Status-Code 401 (UNAUTHORIZED) und einer Fehlermeldung abgelehnt.

4.6 Screening Service

4.6.1 Frontend

Mit dem Frontend des Screeningservices kann das Backend des Communication Services konfiguriert werden.

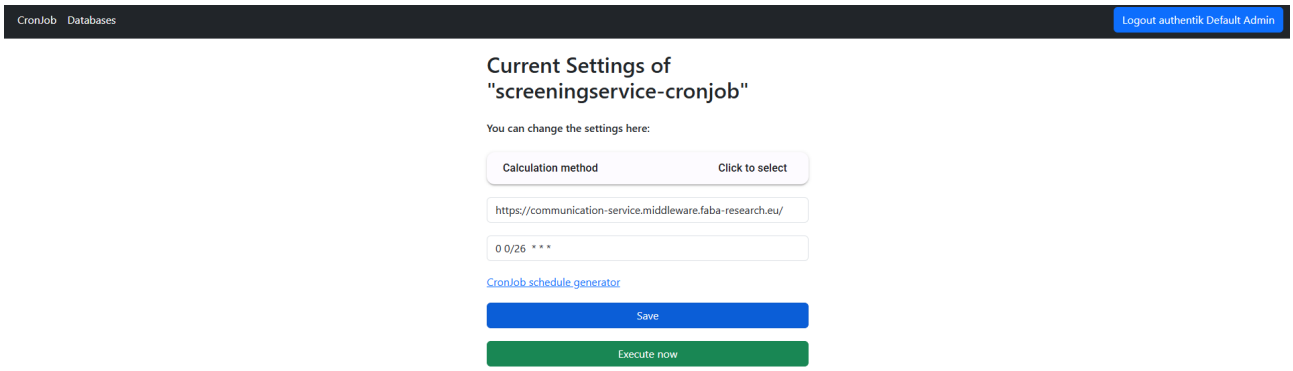


Abbildung 24: Screening-Service: Frontend

4.6.1.1 Konfiguration

Die Konfiguration der Umgebungsvariablen erfolgt genau demselben Schema wie des Communication-Services. Dieses wird in der Section *Konfiguration* des Communication-Services erklärt.

4.6.1.2 Authentifizierung

Die Authentifizierung des Screening-Services erfolgt auf die gleiche Weise wie im Communication-Service über Authentik und mit dem NPM-Package *angular-oauth2-oidc*. Eine genaue Erklärung befindet sich in der Section *Authentifizierung* des Communication-Services.

4.6.1.3 Aufbau

Der Aufbau des Screening-Service Frontends ähnelt stark dem Frontend des Communication-Services. Am oberen Rand ist eine Navigationsbar zu finden, mit der man durch das Frontend navigieren kann. In der Navigationsbar befindet sich ein Logout Button, mit dem man den aktuellen Benutzer abmelden kann. Die Konfigurationsseite *CronJob* beinhaltet eine Möglichkeit, Berechnungsmethoden auszuwählen, die dann durch den Screening-Service ausgeführt werden. Die Berechnungsmethoden befinden sich in einem *Expansion Panel*, das von Angular Material zur Verfügung gestellt wird. Genauer erklärt wird dies in der Section *Design*. Unter diesem Expansionpanel befinden sich zwei Eingabefelder. Im ersten Eingabefeld, wird der Link zum Communication-Service eingegeben und im zweiten Eingabefeld, wird eine *CronJob schedule expression* eingegeben. Mit dem Button *Save* können die Einstellungen gespeichert werden und dem Backend geschickt werden. Mit *Execute now* kann der CronJob manuell gestartet werden.

Die Konfigurationsseite *Databases* ist exakt gleich aufgebaut mit der Datenbankkonfiguration im Communication-Service: *Aufbau*

4.6.1.4 Design

Im Screening-Service Frontend wurde sowie im Communication-Service *Bootstrap* Frontend verwendet, um ein einheitliches Design sicherzustellen. Neben Bootstrap wurde zusätzlich *Angular Material* verwendet, um Dinge wie: *Schalter oder Expansion Tables* einfachst in das Frontend zu integrieren. [77] [80]

4.6.2 Backend

Das Backend vom Screening-Service besteht aus zwei wesentlichen Teilpunkten: Der Cronjob, welcher die Berechnungen ausführt, sowie eine API, um diesen Cronjob zu verwalten. Der Screening-Service ist in einem Projektverzeichnis implementiert, die API und der Cronjob werden allerdings separat im Kubernetes-Cluster deployed.

4.6.2.1 Database-Utilities

Das Modul `database_utils.py` ist ein Teil des Screening-Services und dient dazu, Datenbankoperationen zu verwalten. Darüber hinaus sind in diesem Modul auch Funktionen implementiert, um die Berechnungsmethoden zu verwalten.

```
from pymongo import MongoClient
from src.config.screening_config import mongodb_url

client = MongoClient(mongodb_url["url"])

sensorDB = client['sensor_data']
screeningserviceSettingsDB = client['screening_service_settings']
statusCollection = screeningserviceSettingsDB['calculations_status']
```

PyMongo wird verwendet, um die Verbindung zur MongoDB zu verwalten.

- `sensorDB` speichert die Daten.
- `screeningserviceSettingsDB` speichert Konfigurationen im Bezug auf den Screening-Service-Cronjob
- `statusCollection` ist eine Collection in den Screening-Service-Einstellungen zur Verwaltung der Berechnungsmethoden.

```
def findCalculationMethod(calculation_method_name):
    statusCollection = screeningserviceSettingsDB['calculations_status']
    calculationMethod = statusCollection.find_one({"calculation":
    ↪  calculation_method_name})
    return calculationMethod
```

Die Funktion `findCalculationMethod` wird genutzt, um eine bestimmte in der Datenbank gespeicherte Berechnungsmethode anhand dessen Namen abzurufen.

```
def toggleCalculationMethod(calculation_method_name):
    statusCollection = screeningserviceSettingsDB['calculations_status']
    calculation = findCalculationMethod(calculation_method_name)
    if calculation:
        new_status = not calculation['enabled']
        result = statusCollection.update_one(
```

```
            {"calculation": calculation_method_name},
            {"$set": {"enabled": new_status}}
        )
        if result.matched_count == 0:
            return calculation_not_found_error
        return {"success": "Status updated successfully"}
    else:
        return calculation_not_found_error
```

Die Funktion `toggleCalculationMethod` verändert den Status `enabled` der Berechnungsmethode, und speichert diesen neu in der Datenbank. Dabei wird die Funktion `findCalculationMethod` genutzt, um die Berechnungsmethode aus der Datenbank abzurufen. Falls die Berechnungsmethode, dessen Status verändert werden soll, nicht gefunden wird, wird ein Fehler zurückgegeben.

```
def getAllCalculations():
    calculations_cursor = statusCollection.aggregate([{"$project": {"_id":
    ↪  0}}])
    return list(calculations_cursor)
```

Die Funktion `getAllCalculations()` gibt eine Liste aller in der Datenbank gespeicherten Berechnungsmethoden zurück. Sie wird in der Screening-Service-API genutzt.

Berechnung von Luftfeuchtigkeitsdaten

Der Screening-Service wurde um eine weitere Berechnungsmethode erweitert. Dabei handelt es sich um die Berechnung von täglichen Statistiken über eingegangene Luftfeuchtigkeitsdaten, konkret den Mininal- und Maximalwert, sowie den Durchschnittswert und die Standardabweichnung. Die Berechnung findet in der Funktion `calculateHumidityTrends()` statt. Ein Beispiel eines gespeicherten Feuchtigkeitswert sieht folgendermaßen aus:

```
{
  "_id": "67949e924664e7896f5c69f1",
  "timestamp": "2024-02-25",
  "value": 55.2,
  "description": "humidity",
  "unit": "percent",
}
```

Die Berechnung der Statistiken erfolgt mithilfe einer MongoDB-Aggregation-Pipeline, welche auf die `sensor_humidity`-Collection der Datenbank angewandt wird.

Da das Datum in der Datenbank als String gespeichert wird, wird es im ersten Schritt der Pipeline mit `$dateFromString` in ein Datumsformat umgewandelt, um damit rechnen zu können. Im zweiten Schritt der Pipeline werden die Werte nach Tag, Monat, und Jahr gruppiert, um alle Messungen eines einzelnen Tages zusammenzufassen. Anschließend wird der Minimalwert mit `$min`, der Maximalwert mit `$max`, der Durschnittswert mit `$avg` und die Standardabweichung mit `$stdDevSamp` berechnet. Im nächsten Schritt der Pipeline werden der Durchschnittswert und die Standardabweichung mit `$round` auf zwei Nachkommastellen gerundet, um eine bessere Lesbarkeit zu gewährleisten. Im vierten Schritt wird das im ersten Schritt umgewandelte Datumsobjekt wieder mittels `$dateToString` zurück in einen String umgewandelt. Abschließend werden die berechneten Statistiken aufsteigend nach Datum sortiert. Nachdem die Pipeline definiert wurde, wird sie mittels `sensorDB['sensor_humidity'].aggregate(pipeline)` auf der Datenbank angewandt, um die Berechnung auszuführen. Die Ergebnisse der Berechnung werden im `results`-Objekt zurückgegeben. [81] Ein Beispielsatz für das `results`-Objekt sieht folgendermaßen aus:

```
{
  "date": "2024-02-25",
  "min_value": 53.0,
  "max_value": 57.0,
  "average_value": 55.01,
  "standard_deviation": 2.35
}
```

4.6.2.2 API

Kubernetes Utilities

Das Modul `kubernetes_utils.py` ist ein Teil des Screening-Services und enthält Funktionen für die API, welche den Zugriff auf Kubernetes ermöglichen. Primär stellt es Funktionen für die Verwaltung des Cronjobs und die Aktualisierung der Communication-Service-URL bereit. Um die Interaktion mit Kubernetes möglich zu machen, verwendet das Modul die Kubernetes-API-Clientbibliothek.

```
config.load_incluster_config()


v1 = client.BatchV1Api()
core_v1 = client.CoreV1Api()
```

Die Methode `config.load_incluster_config()` lädt die Konfigurationen des Kubernetes-Clusters. Des Weiteren werden zwei API-Clients erstellt: `v1` für die Cronjobs, und `core_v1` für die Verwaltung von Secrets.

```
def run_cronjob(namespace, cronjob_name):
    cronjob = v1.read_namespaced_cron_job(name=cronjob_name,
    ↪  namespace=namespace)
    job_name = f"{cronjob_name}-{datetime.now().strftime('%Y%m%d%H%M%S')}-{u
    ↪  uid.uuid4().hex[:5]}"
    job = client.V1Job(
        metadata=client.V1ObjectMeta(name=job_name),
        spec=client.V1JobSpec(
            template=cronjob.spec.job_template.spec.template,
        )
    )
    response = v1.create_namespaced_job(namespace=namespace, body=job)
    return {"success": "Cronjob started successfully"}
```

Die Funktion `run_cronjob` startet einen neuen Cronjob basierend auf einer vorhandenen Cronjob-Vorlage. Dafür sind die Parameter `namespace`, also der Namespace, in dem sich die Cronjob-Vorlage befindet, sowie `cronjob_name`, welcher den Namen der Cronjob-Vorlage repräsentiert, erforderlich. Um den neuen Cronjob einzigartig identifizierbar zu machen, wird ein Name aus mehreren Bestandteilen generiert:

- Zuerst der Name der Cronjobvorlage,
- im Anschluss ein Zeitstempel
- am Ende eine zufällig generierte UUID mithilfe der Python-Bibliothek **uuid**

Im Anschluss wird ein `V1Job`-Objekt erzeugt, welches das `job_template`, also die Spezifikationen, der Cronjob-Vorlage nutzt. Der Job wird dann mit `v1.create_namespaced_job(namespace=namespace, body=job)` an Kubernetes übergeben und dort ausgeführt. Bei erfolgreichem Abschluss wird eine Erfolgsnachricht zurückgegeben, im Fehlerfall eine entsprechende Fehlermeldung.

Eine weitere Funktion, `retrieve_cronjob_schedule` ermöglicht das Abrufen des aktuellen Cronjob-Intervalls. Als Übergabeparameter werden `namespace` und `cronjob_name` benötigt. Mit `v1.read_namespaced_cron_job(name=cronjob_name, namespace=namespace)` wird der Cron-

job aus dem Kubernetes-Cluster abgerufen, und das Ergebnis wird dem `cronjob`-Objekt zugewiesen. Am Ende wird `cronjob.spec.schedule` zurückgegeben.

```
def update_cronjob_schedule(namespace, cronjob_name, new_schedule):
    cronjob = v1.read_namespaced_cron_job(name=cronjob_name,
    ↪  namespace=namespace)
    cronjob.spec.schedule = new_schedule
    v1.patch_namespaced_cron_job(name=cronjob_name, namespace=namespace,
    ↪  body=cronjob)
    return {"success": "Cronjob Schedule updated successfully."}
```

Mithilfe der Funktion `update_cronjob_schedule` kann man das Intervall des Cronjobs aktualisieren. Zusätzlich zu den Parametern `namespace` und `cronjob_name` muss man nun auch den Parameter `new_schedule` angeben. Das Intervall muss in der Cron-Syntax angegeben werden. In der oben gezeigten Funktion wird zuerst der aktuelle Cronjob aus dem Kubernetes-Cluster geladen. Im Anschluss wird dem Cronjob-Objekt das neue Schedule zugewiesen. Mit `v1.patch_namespaced_cron_job(name=cronjob_nam` `namespace=namespace` `, body=cronjob)` wird der Cronjob mit dessen neuen Schedule im Kubernetes-Cluster aktualisiert.

Die Funktion `list_cronjobs(namespace)` dient dazu, alle Namen der Cronjobs in einem gegebenem Namespace zurückzugeben. Dazu ist die Angabe des Übergabeparameters `namespace` notwendig. Zuerst wird in das `cronjobs`-Objekt eine Liste der Cronjobs mittels `v1.list_namespaced_cron_job(namespace=namespace)` geladen. Im Anschluss wird mit `return [cronjob.metadata.name for cronjob in cronjobs.items]` eine Liste mit den Cronjob-Namen zurückgegeben.

```
def retrieve_communication_service_url(namespace, secret_name):
    secret = core_v1.read_namespaced_secret(name=secret_name,
    ↪  namespace=namespace)
    return b64decode(secret.data['API_URL']).decode('utf-8')
```

Die Funktion `retrieve_communication_service_url` ermöglicht das Abrufen der aktuellen Communication-Service-URL. Als Parameter muss man erneut `namespace`, sowie `secret_name` angeben. `secret_name` repräsentiert den Namen des Secret-Files im Kubernetes-Cluster. Zuerst wird das Secret mittels `core_v1.read_namespaced_secret(name=secret_name, namespace=namespace)` ausgelesen. Da Kubernetes-Secrets Base64-kodiert gespeichert sind, wird die URL mit der Funktion

b64decode dekodiert und anschließend in utf-8 umgewandelt. Zurückgegeben wird schlussendlich die entschlüsselte URL zurück.

```
def set_communication_service_url(namespace, secret_name, new_api_url):
    secret = core_v1.read_namespaced_secret(name=secret_name,
    ↪  namespace=namespace)
    secret.data['API_URL'] =
    ↪  b64encode(new_api_url.encode('utf-8')).decode('utf-8')
    core_v1.patch_namespaced_secret(name=secret_name, namespace=namespace,
    ↪  body=secret)
    return {"success": "Communication-Service-url updated successfully."}
```

In dieser Funktion kann man die URL des Communication-Services aktualisieren. Zusätzlich zu den Parametern namespace und secret_name muss man nun auch den Parameter new_api_url angeben. Die neue API-URL wird in das Base64-Format kodiert. Mit dem Methodenaufruf core_v1.patch_namespaced_secret(name=secret_name, namespace= namespace, body=secret) wird diese Secret-Änderung im Kubernetes-Cluster übernommen.

API-Endpoints

Die API-Endpoints selbst sind in api.py implementiert. Sie sind also die zentrale Komponente, welche die Funktionen aus den vorher erklärten Modulen verwendet und für die Implementierung der Endpoints heranzieht. Für den Request-Body werden außerdem Models verwendet, um die Daten zu lesen und zu validieren. Die API wird mittels FastAPI implementiert. Alle nachfolgend gezeigten Endpoints geben im Fehlerfall eine HTTP-Exception mit einem Statuscode zurück.

```
class CronjobData(BaseModel):
    namespace: str
    cronjob_name: str


@app.post("/cronjob")
async def post_start_cronjob(data:CronjobData, user: str =
↪  Depends(verify_token)):
    return run_cronjob(data.namespace, data.cronjob_name)
```

Der POST-Endpoint /cronjob ermöglicht das Starten eines Cronjobs basierend auf einer Cronjob-Vorlage. In der Request wird im Body der Namespace, in dem sich die Vorlage befindet, sowie der Name der Vorlage mitgegeben. Dieser Endpoint, sowie auch alle nachfolgend erläuterten, werden

mithilfe der `verify_token`-Funktion des Authentication-Modules abgesichert. Zur Ausführung des Cronjobs wird die `run_cronjob`-Funktion des Kubernetes-Utilities-Moduls verwendet.

```
class UpdateCronjobScheduleData(BaseModel):
    namespace: str
    cronjob_name: str
    new_schedule: str


@app.post("/cronjob-schedule")
async def post_update_cronjob_schedule(data:UpdateCronjobScheduleData, user:
↪   str = Depends(verify_token)):
    return update_cronjob_schedule(data.namespace, data.cronjob_name,
    ↪   data.new_schedule)
```

Der POST-Endpoint `/cronjob-schedule` dient dazu, das Intervall des Cronjobs zu aktualisieren. Dazu ist es neben dem Namespace und dem Namen des Cronjobs auch notwendig, das neue Intervall im Cron-Format im Body mitzugeben. Zur Aktualisierung wird die `update_cronjob_schedule`-Funktion des Kubernetes-Utilities-Moduls genutzt. Da der Cronjob-Schedule aktualisiert wird, ist hierfür ein PUT-Endpoint geeigneter.

```
class ToggleCalculationRequest(BaseModel):
    calculation: str


@app.post("/toggle-calculation")
async def toggle_calculation(request: ToggleCalculationRequest, user: str =
↪   Depends(verify_token)):
    return toggleCalculationMethod(request.calculation)
```

Der POST-Endpoint `/toggle-calculation` wird genutzt, um eine Screening-Service-Berechnungsmethode ein- oder ausschalten. In der Anfrage wird der Name der Berechnungsmethode mitgegeben, welcher dann in der `toggleCalculationMethod`-Funktion des Database-Utilities-Moduls mitgegeben wird. Da eine bestehende Berechnungsmethode aktualisiert wird, ist ein PUT-Endpoint eine bessere Wahl.

```
class CommunicationServiceUrlData(BaseModel):
    namespace: str
    secret_name: str
    new_api_url: str
```

```
@app.post("/communication-service-url")
async def change_communication_service_url(data:CommunicationServiceUrlData,
↪  user: str = Depends(verify_token)):
    return set_communication_service_url(data.namespace, data.secret_name,
    ↪  data.new_api_url)
```

Mit dem POST-Endpoint `/communication-service-url` ist es möglich, die URL des Communication-Services zu ändern. Diese URL ist in einem Kubernetes-Secret gespeichert. In der Request werden daher der Namespace und der Name des Secret-Files samt der neuen URL mitgegeben. Zur Aktualisierung wird die Funktion `set_communication_service_url`-Funktion des Kubernetes-Utilities-Modul herangezogen. Auch für diesen Endpoint wäre ein PUT-Endpoint geeigneter, da die URL aktualisiert wird.

```
@app.get("/screening-service-data", description="Retrieve screening service
↪  data")
async def get_screening_service_data(secret_name: str, namespace: str,
↪  cronjob_name: str, user: str = Depends(verify_token)):
combined_data = {}

communication_service_url_result =
↪  retrieve_communication_service_url(namespace, secret_name)
combined_data["communication_service_url"] = communication_service_url_result

cronjob_schedule_result = retrieve_cronjob_schedule(namespace, cronjob_name)
combined_data["cronjob_schedule"] = cronjob_schedule_result

cronjobs_result = list_cronjobs(namespace)
combined_data["cronjobs"] = cronjobs_result

calculations_result = getAllCalculations()
combined_data["calculations"] = calculations_result

return combined_data
```

Der GET-Endpoint `/screening-service-data` liefert ein Dictionary aus verschiedenen Informationen zurück. Bei einer Anfrage werden `namespace`, `secret_name` und `cronjob_name` als Query-Parameter mitgegeben. Falls ein Fehler beim Abruf der Informationen auftritt, wird ein Fehler als Response zurückgegeben. Zurückgegeben werden folgende Daten:

- Die aktuelle Communication-Service-URL,
- Das aktuell in Kubernetes gespeicherte Cronjob-Intervall.
- Alle Cronjobs in einem Namespace.
- Alle aktuell in der Datenbank gespeicherten Berechnungsmethoden.

Diese Daten werden im Dictionary `combined_data` zusammengestellt, damit der Zugriff auf diese Daten im Frontend erleichert wird.

4.6.2.3 Cronjob

Der Screening-Service-Cronjob führt die gespeicherten Berechnungmethoden basierend auf den in der Datenbank gespeicherten Daten aus. Dafür nutzt er unter anderem auch Funktionen aus dem Database-Utilities-Modul. Bei der Ausführung des Cronjobs wird das Modul `screening_service.py` ausgeführt.

Das Modul `screening_service.py` enthält die Funktion `createHumidityInformation`, welche das Ergebnis der Screening-Service-Berechnungen als Übergabeparameter erwartet. Diese Funktion stellt dann dynamisch ein Objekt zusammen, welches die Attribute `objexternalkey`, `infoshortdescription` und `infodescription` enthält. In die Werte dieser Attribute werden dann die Daten der berechneten Werte eingefügt. `infoshortdescription` enthält eine kurze Beschreibung der neuen Nachricht, und `infodescription` ist eine detailliertere Beschreibung. Die fertig zusammengebaute Nachricht wird dann an das Communication-Service-Backend weitergeleitet.

Speicherung der berechneten Luftfeuchtigkeits-Statistiken

```
def saveHumidityTrends():
    results = calculateHumidityTrends()

    for result in results:
        check = humidityTrendsDB['daily_humidity_trends'].find(result)
        leng = len(list(check.clone()))
        if leng == 0:
            info = createHumidityInformation(result)
```

```
            saveDocument('daily_humidity_trends', result)
            sendToCAQ(info)
        else:
            print('day trends already saved')
```

Die Funktion `saveHumidityTrends()` ruft zuerst die in Abschnitt 4.6.2.1 näher erklärte Berechnungsmethode `calculateHumidityTrends()` aus dem Database-Utils-Modul aus, und prüft im Anschluss, ob die Berechnung eines bestimmten Tagessatzes bereits vorher in der Datenbank gespeichert wurden. Falls bereits eine Berechnung des Tages existiert, wird sie nicht erneut gespeichert. Falls von diesem Tag noch keine Berechnungen gespeichert wurden, werden sie nun mit der `saveDocument`-Funktion in der Datenbank gespeichert, und leitet mit dem Funktionsaufruf `sendToCAQ(info)` die Informationsnachricht über die neuen Berechnungen an das Communication-Service-Backend weiter.

Verwaltung der Berechnungsmethoden

```
calculation_methods = {
    "humidity_trends_calculation": {
        "method": saveHumidityTrends,
        "enabled": True
    },
    "outliers_calculation": {
        "method": calculateOutliers,
        "enabled": True
    }
}
```

Die Berechnungsmethoden werden in einem Objekt `calculation_methods` im Cronjob definiert. Jede Berechnungsmethode hat einen Namen, eine zugehörige Funktion und ihren Status. Falls eine Berechnungsmethode in dieser Definition noch nicht in der Datenbank gespeichert wurde, erfolgt auch die Speicherung im Cronjob.

Ausführung der Berechnungen

```
for calculation_name in calculation_methods:
    db_calculation = findCalculationMethod(calculation_name)
    if db_calculation and db_calculation['enabled']:
        calculation_methods[calculation_name]['method']()
```

Die letzte Aktion des Cronjobs ist schlussendlich die Ausführung aller aktiven Berechnungsmethoden.

4.6.2.4 Tests

Mit der Python-Bibliothek `pytest` sind Unit-Tests für die Screening-Service-API implementiert, um die Robustheit der API sicherzustellen. Für das Senden der HTTP-Anfragen wird die `requests`-Bibliothek verwendet. Die `httpx`-Bibliothek wird verwendet, um die Status-Codes zu vergleichen.

Beispiel: GET-Endpoint `/screening-service-data`

```
test_params_screening_service_data_success = {
    "secret_name": "caqcredentials",
    "namespace": "kiramet-middleware",
    "cronjob_name": "screeningservice-cronjob"
}
def test_get_screening_service_data_success():
    result = requests.get(screening_api_url+"/screening-service-data",
    ↪  test_params_screening_service_data_success)
    assert result.status_code == httpx.codes.OK
```

In diesem Test wird eine Anfrage mit gültigen Parametern gesendet, und es wird eine Antwort mit dem Status Code 200 (OK) erwartet. Für jeden Endpoint gibt es einen Test mit gültigen Parametern, und weitere Tests, welche folgende Szenarien abdecken:

- Es wird getestet, ob alle Parameter vorhanden und gültig sind für eine Request, andernfalls wird der Status-Code 422 (UNPROCESSABLE ENTITY) zurückgegeben.
- Es wird überprüft, ob Resourcen im Kubernetes Cluster, wie zum Beispiel der Cronjob, existieren. Andernfalls wird der Status-Code 404 (NOT FOUND) zurückgegeben.
- Es wird getestet, ob die Berechtigung auf eine Resource zuzugreifen vorhanden ist, andernfalls wird der Status Code 403 (FORBIDDEN) zurückgegeben.

4.7 MongoDB-UI

Die MongoDB-Datenbank im System speichert alle Sensor- und Analyse-Daten. MongoDB-UI ermöglicht einen Einblick in die Daten, die diese Datenbank speichert. MongoDB-UI besteht aus einem Angular-basierten Frontend, welches in Abbildung 25 zu sehen ist und einem Python-basierten Backend. Die folgenden Unterkapitel beschreiben die Implementierung der Webanwendung MongoDB-UI.

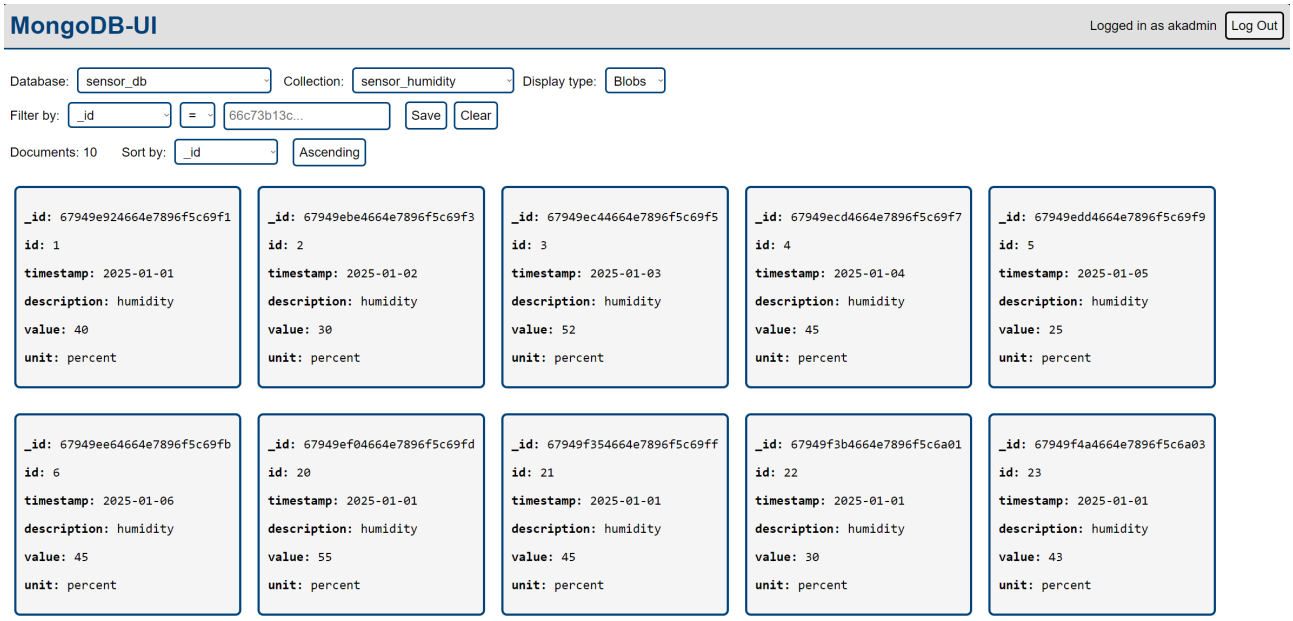


Abbildung 25: MongoDB-UI Website

4.7.1 Backend

Das Backend ist eine Python-Applikation, die Daten aus der MongoDB-Datenbank ausliest und über eine geschützte REST-API zur Verfügung stellt. Das Starten der Applikation erfolgt durch das Ausführen der app.py-Datei mit Python. Diese Datei beinhaltet die Definition der REST-API und stellt diese mit der Bibliothek *uvicorn* zur Verfügung.

4.7.1.1 Datenbankzugriff

Der Datenbankzugriff erfolgt in einer eigenen Python-Datei namens *dbConnection.py*, welche die Bibliothek *pymongo* nutzt, um eine Verbindung mit der Datenbank aufzubauen und Daten aus dieser auszulesen.

Verbindungsaufbau

Die Verbindung zur Datenbank wird mit der *MongoClient*-Klasse hergestellt. Diese Klasse benötigt hierfür einen *Connection String*. Dieser Connection String wird mit der Bibliothek *os* aus den Umgebungsvariablen ausgelesen. Hierfür wird die Methode os.getenv() verwendet, welche nach einer Umgebungsvariable mit dem Schlüssel *mdbURL* sucht. Sollte in der Umgebung, in der die Applikation gestartet wird, keine Umgebungsvariable mit dem Schlüssel *mdbURL* vorhanden sein, wird ein Standard-Wert verwendet. Das ist immer dann der Fall, wenn die Applikation in einer Entwicklungsumgebung gestartet wird. Der folgende Code zeigt, wie dieser Vorgang implementiert wird:

```
connectionString = os.getenv("mdbURL","mongodb://localhost:27017/")
client = MongoClient(connectionString)
```

Die *client* Variable wird im weiteren Code der *dbConnection.py*-Datei verwendet, um auf die Datenbank zuzugreifen.

Auslesen der Daten

Das Auslesen der Daten erfolgt in drei Funktionen, welche dann von der REST-API verwendet werden. Die erste Methode *getDBs()* benötigt keine Übergabeparameter und gibt die Namen aller relevanten Sub-Datenbanken in der MongoDB-Datenbank als String Liste zurück. Um alle die Namen zu erhalten, wird die Methode *client.list_database_names()* verwendet. Diese Methode gibt eine Liste aller Datenbank-Namen der Datenbank, mit der der Client eine Verbindung aufgebaut hat, in Form einer String Liste zurück. Bevor die Liste zurückgegeben wird, werden noch die Sub-Datenbanken *admin*, *local* und *config* aus der Liste entfernt, da sie nur MongoDB-spezifische Daten enthalten, welche für den Anwendungsfall der MongoDB-UI irrelevant sind. Der folgende Code beschreibt die Definition der *getDBs()* Funktion:

```
def getDBs():
    dbs = client.list_database_names()
    #remove mongodb-specific data
    dbs.remove("admin")
    dbs.remove("local")
    dbs.remove("config")
    return dbs
```

Die zweite Funktion *getColls()* benötigt einen Übergabeparameter und gibt die Namen aller Collections einer Sub-Datenbank als String Liste zurück. Als Übergabeparameter erwartet die Funktion den Namen einer Sub-Datenbank, welche innerhalb der MongoDB-Datenbank existiert. Die Funktion

überprüft, ob der übergebene Datenbankname zu einer existierenden Sub-Datenbank passt, indem sie die *getDBs()* Funktion aufruft und dann überprüft, ob der übergebene Datenbankname in der Liste von Datenbank-Namen enthalten ist. Sollte der übergebene Datenbankname zu keiner existierenden Sub-Datenbank passen, wird eine leere Liste zurückgegeben. Der folgende Code beschreibt die Definition der soeben erklärten Überprüfung:

```
if not dbs.__contains__(dbName):
    return []
```

Nach der Überprüfung wird die Methode *client[dbName].list_collection_names()* verwendet um eine Liste aller Collections der spezifizierten Sub-Datenbank zu erhalten. Da die *client*-Variable nur die allgemeine Verbindung zur MongoDB-Datenbank beinhaltet, muss man die Sub-Datenbank, aus welcher die Collections ausgelesen werden sollen, mit einem Schlüssel-basierten Zugriff angeben. Die Liste der Collection-Namen wird dann direkt zurückgegeben. Im folgendem Code ist zu sehen wie die *list_collection_names()*-Methode verwendet wird.

```
colls = client[dbName].list_collection_names()
```

Die dritte Funktion *getDocuments()* benötigt zwei Übergabeparameter und gibt alle Dokumente einer Collection einer Sub-Datenbank als Objekt Liste zurück. Als Übergabeparameter erwartet die Funktion den Namen einer Sub-Datenbank, welche innerhalb der MongoDB-Datenbank existiert, und den Namen einer Collection, welche innerhalb der übergebenen Sub-Datenbank existiert. Um zu überprüfen ob die Sub-Datenbank existiert, wird der selbe Code wie in der *getColls()* Funktion verwendet. Auch die Existenz der übergebenen Collection innerhalb der spezifizierten Sub-Datenbank wird überprüft. Hierfür wird das selbe Prinzip verwendet, wie beim Überprüfen auf die Existenz der Sub-Datenbank. Statt der *getDBs()* Funktion wird aber die *getColls()* Funktion verwendet um die Collection-Namen für den Vergleich zu erhalten. Sollte der übergebene Datenbankname oder der übergebene Collectionname zu keiner existierenden Sub-Datenbank oder Collection innerhalb der spezifizierten Sub-Datenbank passen, wird eine leere Liste zurückgegeben. Der folgende Code beschreibt wie die Existenz der übergebenen Collection überprüft wird.

```
if not colls.__contains__(collName):
    return []
```

Nach der Überprüfung wird ein Verweis auf die Collection in eine Lokale Variable kopiert. Hierfür wird ein doppelter Schlüssel-basierter Zugriff verwendet um auf die Sub-Datenbank und dann auf die Collection zuzugreifen. Der folgende Code zeigt wie auf die Collection zugegriffen wird:

```
coll = client[dbName][collName]
```

Aus der *coll* Variable werden dann durch die *.find()*-Methode alle gespeicherten Dokumente der Collection aufgerufen. Da der Rückgabewert der *.find()*-Methode ein *Cursor* ist und keine Liste, muss man diesen Umwandeln. Ein Cursor ist ein iterierbares Objekt welches die Dokumente speichert. Da er iterierbar ist, kann man mit einer *List Comprehension* den Cursor umwandeln. Die Liste die aus der *List Comprehension* gebildet wird, wird dann direkt von der Funktion zurückgegeben. Der folgende Code definiert den soeben erklärten Bestandteil der *getDocs()* Funktion:

```
documents = coll.find()
document_list = [doc for doc in documents] # <- List Comprehension
return document_list
```

4.7.1.2 API-Definition

Das Backend stellt eine REST-API zur Verfügung, welche mittels FastAPI implementiert wurde. Die API wird nach dem selben Schema, wie dem in Sektion 4.4, implementiert und nutzt die selben Konfigurationen. Die gesamte API ist in einer app.py-Datei definiert.

Die API stellt drei Endpoints, jeweils einen für jede Funktion aus der *dbConnection*-Datei, zur Verfügung. Alle drei Endpoints hören auf GET-Requests und sind mit dem Authentication-Module abgesichert.

Der erste Endpoint ist unter dem Pfad */DBNames* erreichbar und gibt die Liste von Datenbanknamen zurück, welche die *getDBs()*-Funktion aus der *dbConnection*-Datei zurückgibt.

Der zweite Endpoint ist unter dem Pfad */CollNames/dbname* erreichbar und erwartet den Namen einer Datenbank als Pfad-Parameter. Mit diesem Datenbanknamen wird dann die *getColls()*-Funktion aus der *dbConnection*-Datei aufgerufen. Die Liste von Collection-Namen, die diese Funktion zurückgibt, wird dann auch direkt vom Endpoint zurückgegeben.

Der dritte Endpoint ist unter dem Pfad */Documents/dbname/collname* erreichbar und erwartet den Namen einer Datenbank und einer Collection als Pfad-Parameter. Mit diesen beiden Parametern wird dann die *getDocs()*-Funktion aus der *dbConnection*-Datei aufgerufen. Da der Datentyp des *_id* Attributs, welches jedes Dokument hat, nicht direkt serialisiert werden kann, wird die Dokumenten-Liste, welche die *getDocs()*-Funktion zurück gibt vor dem weiteren zurückgeben noch in einer for-Schleife überarbeitet. Innerhalb dieser for-Schleife werden die hexadezimal-IDs, welche in den *_id* Attributen gespeichert werden, in einen normalen String umgewandelt und dem *_id* Attribut neu zugewiesen. Der folgende Code implementiert die soeben erklärte Umwandlung:

```
for doc in documents:
        if "_id" in doc:
            doc["_id"] = str(doc["_id"])
```

Nach dieser Umwandlung wird die Liste zurückgegeben.

Da viele Deserialisierungs-Tools nicht mit direkten JSON-Listen arbeiten können, sondern ein JSON-Objekt erwarten, werden alle Listen, vor dem zurückgeben in einem Objekt mit einem *content* Attribut verpackt. Folgender Code zeigt diesen Vorgang anhand der Dokumenten-Liste:

```
response = {"content": documents}
return response
```

Die API wird standardmäßig auf Port 5000 gestartet.

4.7.1.3 Deployment

Das Deployment des MongoDB-UI-Backends erfolgt nach dem selben Prinzip, wie es in der Sektion 4.1 erklärt wird. Das Dockerfile beinhaltet als zusätzliche Umgebungsvariable den Connection-String der MongoDB-Datenbank.

4.7.1.4 Tests

Es existieren mehrere Tests, die das erfolgreiche Auslesen aus der MongoDB-Datenbank überprüfen. Diese Tests sind in einer test_all.py Datei definiert und werden mit der python-internen *unittest*-Bibliothek durchgeführt.
Innerhalb der test_all.py-Datei ist eine *TestDBConnection*-Klasse definiert, die von der *unittest.TestCase*-Klasse erbt. Diese Klasse beinhaltet alle Testfälle, wovon es insgesamt drei gibt.

Die drei Tests überprüfen, ob die drei Funktionen *getDBs()*, *getColls* und *getDocs()* korrekt funktionieren. Dafür werden die Funktionen aufgerufen, woraufhin überprüft wird, ob die Liste von Werten, die sie zurück geben, nicht leer sind.

Der Test für die *getColls()* Funktion ist beispielsweise wie folgt definiert:

```
\textit{def test_getColls(self):
        dbs = getDBs()
        colls = getColls(dbs["content"][0])
        self.assertTrue((colls["content"].__len__()>0))}
```

Damit die *getColls()* Funktion ausgeführt werden kann, wird die *getDBs()* Funktion aufgerufen, um die Datenbank-Namen zu erhalten. Für den Aufruf der *getColls()* Funktion wird dann die erste Sub-Datenbank aus der Liste verwendet. Mit *self.assertTrue()* überprüft der Test, ob der übergebene Vergleich *True* ergibt, wenn ja, ist der Test erfolgreich, wenn nicht, schlägt der Test fehl. Der übergebene Vergleich gibt *True* zurück, wenn die Länge der *colls* Liste, die *getColls()* zurückgibt, länger als null ist.

4.7.2 Frontend

Das Frontend ist eine Angular-Applikation, welche Daten aus der Datenbank über das Backend ausliest und dann visuell aufbereitet. Die Web-Applikation bietet Möglichkeiten zum Sortieren und Filtern der Daten.

4.7.2.1 Bestandteile und deren Aufgaben

Die Web-Applikation nutzt *Angular Components*, um die Webseite in mehrere Bestandteile aufzuteilen. Die Webseite besteht aus folgenden Components:

app: Die App Component beinhaltet die main-view und die loggedOut-view. Die App Component dient als Ausgangspunkt der Component-Aufteilung der Webseite und kümmert sich um das Authentifizierungsverfahren. Sollte ein Nutzer nicht authentifiziert sein, wird die loggedOut-view Component angezeigt. Andernfalls wird die main-view Component angezeigt.

loggedOut-view: Die loggedOut-View Component beinhaltet einen Text, welcher den Nutzer darüber informiert, dass er zurzeit nicht eingeloggt ist. Es ist auch ein Button vorhanden, welcher den Login-Prozess manuell startet. Im Normalfall ist die loggedOut-view Component nur kurz sichtbar, da der Login Prozess automatisch gestartet wird und den Nutzer entweder

zum Identity Provider weiter leitet, oder den Status des Nutzers auf authentifiziert setzt, wodurch die loggedOut-view Component wieder versteckt wird.

main-view: Die main-view Component beinhaltet eine Navigationsleiste und die general-view Component. Die Navigationsleiste ist keine eigene Component und beinhaltet links den Text "MongoDB-UIäls Titel. Weiters wird rechts angezeigt welcher Nutzer gerade eingeloggt ist und rechts davon ist ein Logout Button platziert.

general-view: Die general-view Component beinhaltet die collection-view Component und die Einstellungen für die Auswahl der anzuzeigenden Daten und des Anzeige-Typs.

collection-view: Die collection-view Component zeigt die Dokumente aus der ausgewählten Collection an. Je nach ausgewähltem Anzeige-Typen, werden die Dokumente über eine Auflistung mehrerer document-view Components oder über eine Tabelle angezeigt. Außerdem sind in dieser Component die Filter- und Sortier-Einstellungen implementiert.

document-view: Die document-view visualisiert ein einzelnes Dokument, welches von der collection-view übergeben wird.

In Abbildung 26 ist die Aufteilung der Components auf der Webseite visuell dargestellt.

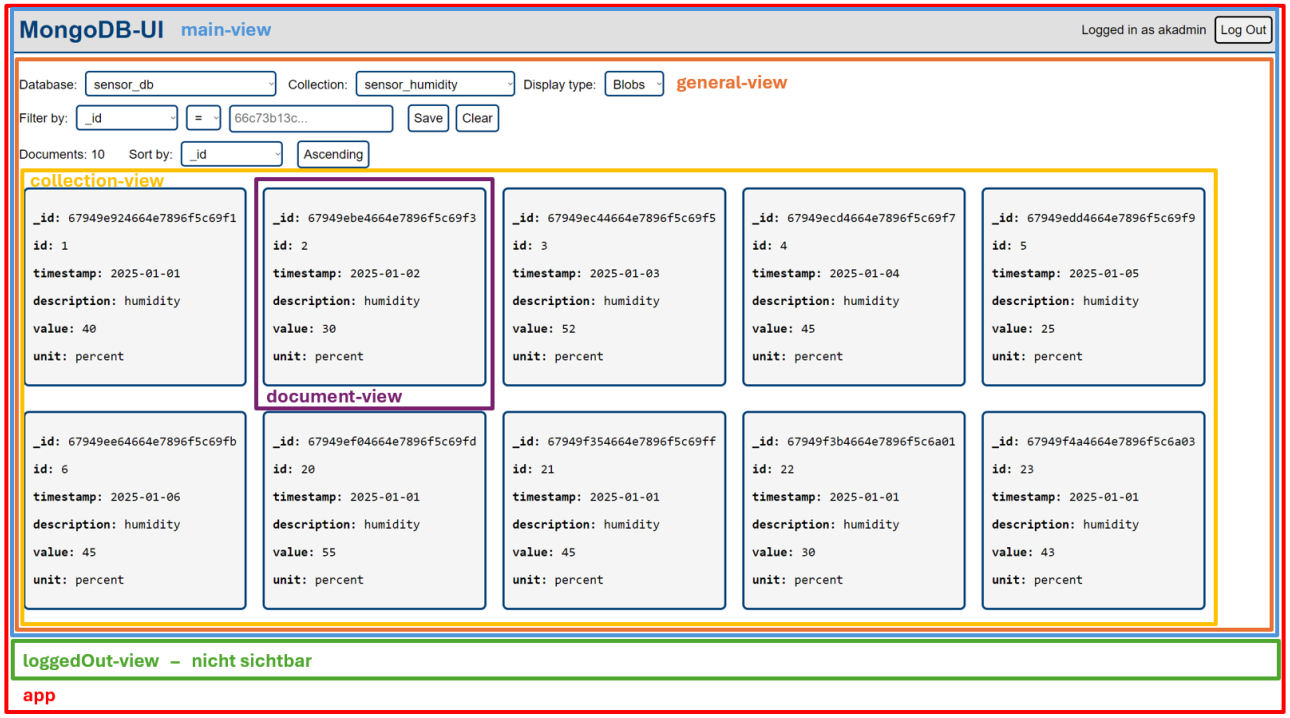


Abbildung 26: Aufteilung der Webseite

4.7.2.2 Authentifizierung

Bevor ein Nutzer auf die Website zugreifen kann, wird überprüft, ob der Client einen Access Token hinterlegt hat. Sollte das nicht der Fall sein, wird der Nutzer zum zentralen Login von Authentik weitergeleitet. Dieser Prozess wird in der App Component durchgeführt. Für diesen Prozess wird die Bibliothek *angular-oauth2-oidc* verwendet. In der AppComponent Klasse wird im Konstruktor eine Instanz der Klasse *OAuthService* mittels *Dependeny Injection* injiziert. Danach wird die *this.authenticate()*-Methode aufgerufen, welche wie folgt definiert ist:

```
authenticate() {
    this.oauthService.configure(authCodeFlowConfig);
    this.oauthService.loadDiscoveryDocumentAndLogin();
    this.oauthService.events
      .pipe(filter((e) => e.type === 'token_received'))
      .subscribe((_) => {
        this.oauthService.loadUserProfile();
    });
}
```

Die Methode *this.oauthService.configure(authCodeFlowConfig)* lädt die Konfigurationen für den Authentifizierungs-Service aus einem Objekt der Klasse *AuthConfig*, welches in diesem Fall *authCodeFlowConfig* heißt. Dieses Objekt wird aus der auth.config.ts-Datei importiert und besitzt mehrere Attribute, welche Informationen für den Kommunikationsaufbau zum Identity Provider beinhalten. Besonders relevant sind die Attribute *issuer* und *clientId*. Beide speichern Daten aus dem *env* Objekt, welches eine Sammlung von Umgebungsvariablen beinhaltet. Das *issuer* Attribut beinhaltet die URL des Identity Providers. Das *clientId* Attribut beinhaltet einen String, der für alle Clients, die auf die Webseite zugreifen, gleich ist. Das *clientId* Attribut ist für den Identity Provider relevant, damit er weiß, dass der User durch MongoDB-UI weitergeleitet wurde.

Nach dem Laden der Konfiguration wird mit der Methode *this.oauthService.loadDiscoveryDocumentAndLogin();* eine Verbindung zum Identity Provider aufgebaut und überprüft, ob der Nutzer sich bereits bei Authentik eingeloggt hat. Sollte das nicht der Fall sein, wird der Nutzer automatisch zum zentralen Authentik Login weitergeleitet.

Parallel zum Login wird mit der Methode *this.oauthService.events.pipe(filter((e) => e.type === 'token_received')).subscribe* ein Listener auf das *token_received*-Event gesetzt. Dieses Event wird ausgelöst, wenn der Login erfolgreich war, oder der Nutzer bereits bei Authentik eingeloggt war und Authentik dem Client den Access Token des Users schickt. Es wird dieser Methode eine weitere

Methode mitgegeben, welche eben dann ausgeführt wird, wenn dieses Event eintritt. Innerhalb der übergebenen Methode wird die Methode *this.oauthService.loadUserProfile();* aufgerufen. Diese Methode sendet eine Anfrage an Authentik für Informationen über den Nutzer, mit dem sich der Client eingeloggt hat.

Auf der Webseite werden alle Funktionalitäten versteckt, bis der Nutzer sich eingeloggt hat. Das erfolgt durch folgendes HTML in der Datei app.component.html:

```
<div *ngIf="hasToken()">
    <app-main-view></app-main-view>
</div>
```

Hier wird die Structural Directive **ngIf* verwendet, um die main-view zu verstecken, welche alle Funktionalitäten der MongoDB-UI beinhaltet. Wenn die *hasToken()*-Methode True zurück gibt, wird die main-view Component angezeigt. Die *hasToken()*-Methode nutzt die *oauthService.hasValidAccessToken()*-Methode um zu überprüfen, ob der Client einen validen Token besitzt und gibt dann True zurück.

Direkt unter dem HTML, welches die main-view Component versteckt, befindet sich ein weiteres div-Element, welches angezeigt wird, wenn die Methode *hasToken()* False zurückgibt. Innerhalb dieses div-Elements befindet sich die loggedOut-view Component.

Außerhalb von der app Component wird auch in der main-view Component die *angular-oauth2-oidc* Bibliothek verwendet. Hier wird beim Laden der Component der Username des eingeloggten Nutzers geladen, um diesen dann anzuzeigen. Hierfür wird wieder die Methode *oauthService.loadDiscoveryDocumentAndLogin()* aufgerufen, um sicherzustellen, dass der Nutzer eingeloggt ist. An diese Methode wird ein *.then()* angehängt, welches wiederum die Methode *oauthService.loadUserProfile()* übergeben bekommt, sodass diese aufgerufen wird, sobald *loadDiscoveryDocumentAndLogin()* abgeschlossen wurde. An die *loadUserProfile()* wird dann wieder ein *.then()* angehängt, welches eine Methode übergeben bekommt, die der lokalen Variable *this.userName* den Username vom Benutzer zuweist, welcher von *loadUserProfile()* geladen wird. Der Code für diese Methodenverkettung schaut wie folgt aus:

```
ngOnInit() {
    this.oauthService.loadDiscoveryDocumentAndLogin().then(()=>{
        this.oauthService.loadUserProfile().then((data)=>{
            let obj: any = data;
            this.userName=obj.info.nickname;
        });
```

```
    });
}
```

Die Variable *this.userName* wird dann in der main-view Component verwendet, um in der Navigationsleiste den Namen anzuzeigen. In dieser Navigationsleiste ist auch einen Logout-Button definiert, welcher die *oauthService.logOut()*-Methode aufruft, wenn er geklickt wird. Die *oauthService.logOut()*-Methode löscht den Access Token aus den Client-Informationen. Dann wird man automatisch zu einer Seite von Authentik weitergeleitet, welche einen darüber informiert, dass man aus der MongoDB-UI-Webseite ausgelogt wurde.

4.7.2.3 Daten einholen

Das Aufrufen der Daten aus dem Backend erfolgt durch einen Service namens *DBConnector*. Im Konstruktor des Service wird eine Instanz vom Angular *HTTPClient* namens *httpClient* und eine Instanz von *OAuthService* namens *oauthService* mittels *Dependency injection* injiziert. Außerdem wird vom *environment* Objekt, welches sich in der environment.ts-Datei befindet das *apiUrl* Attribut verwendet, um der lokalen Variable *API* einen Wert zuzuweisen.

Der *DBConnector* Service stellt drei Methoden zur Verfügung, welche von anderen Components verwendet werden. Außerdem gibt es eine Methode namens *getListData*, welche von den anderen Methoden verwendet wird, diese ist wie folgt definiert:

```
private getListData(path: string): Observable<any> {
    const token = this.oauthService.getAccessToken();
    const headers = new HttpHeaders({
      'Authorization': `Bearer ${token}`
    });

    return this.httpClient.get<any>(this.API + path, { headers });
}
```

Diese Methode übernimmt einen Pfad als Übergabeparameter und holt sich den Access Token vom *oauthService*, um mit diesem dann den *Authorization* Header für die HTTP-Request vorzubereiten. Dann wird mit dem *httpClient* eine GET-Request an eine zusammengebaute URL gesendet. Der Pfad wird mit der Variable *this.API* und dem übergebenem Pfad zusammengebaut. Der Header wird als zusätzlicher Übergabewert mitgegeben. Die Methode gibt den Rückgabewert der *this.httpClient.get()*-Methode direkt wieder zurück. Was in diesem Fall ein *Observable<any>* Objekt ist.

Die drei anderen Methoden des *DBConnector* nutzen die *getListData()*-Methode und heißen *getDBs()*, *getCollections()* und *getDocuments()*. Diese Methoden sind für jeweils eine API-Schnittstelle des Backends gemacht.

Die *getDBs()*-Methode erwartet keine Übergabeparameter und ruft die *getListData()*-Methode mit dem Pfad "DBNamesäuf.

Die *getCollections()*-Methode übernimmt den Namen einer Sub-Datenbank als Übergabeparameter und ruft die *getListData()*-Methode mit einem zusammengesetztem Pfad auf, welcher aus "CollNames/ünd dem übergebenen Datenbank-Namen zusammengebaut wird.

Die *getDocuments()*-Methode übernimmt den Namen einer Sub-Datenbank und einer Collection als Übergabeparameter und ruft die *getListData()*-Methode mit einem zusammengesetztem Pfad auf, welcher aus "Documents/ünd dem übergebenen Datenbank-Namen und Collection-Namen zusammengebaut wird.

4.7.2.4 Anzuzeigende Daten auswählen

Die Funktionalitäten für das Auswählen der anzuzeigenden Daten sind in der general-view Component implementiert. Diese nutzt hierfür auch den *DBConnectorService*. In der general-view Component sind drei Dropdowns implementiert, welche für die Auswahl der Sub-Datenbank, der Collection und des Anzeige-Typen verantwortlich sind.

Das Dropdown für den Anzeige-Typ hat zwei fest vorgegebene Optionen: *List* und *Blobs*. Die beiden Dropdowns für das Auswählen der Sub-Datenbank und Collection wir die Structural Directive **ngFor* verwendet, welche die Optionen auflistet und als Quelle für die Optionen die Listen *databases* und *collections* ausliest. Das folgende HTML zeigt, wie **ngFor* im Dropdown für das Auswählen der Sub-Datenbank verwendet wird:

```
<select class="DBDropdown" id="dbDropdown"
↪  (change)="onDBOptionChange($event)">
    <option *ngFor="let option of databases" [value]="option">
        {{ option }}
    </option>
</select>
```

Beide Listen werden in der *loadData()*-Methode mit Werten gefüllt. Die *loadData()*-Methode wird in der *ngOnInit()*-Methode der general-view Component aufgerufen. Die *loadData()*-Methode ist wie folgt definiert:

```
public async loadData(): Promise<void> {
    this.dbConnectorService.getDBs()
    .subscribe((dataDBs)=>{
        this.databases = dataDBs.content;
        if (this.databases.length != 0) {
            this.selectedDatabase = this.databases[0];
            this.dbConnectorService.getCollections(this.selectedDatabase)
            .subscribe((dataCols)=>{
                this.collections = dataCols.content;
                if (this.collections.length != 0) {
                    this.selectedCollection = this.collections[0];
                    this.updateCollectionView()
                }
            });
        }
    });
}
```

dbConnectorService ist eine Instanz der *DBConnectorService*-Klasse, auf welche in der Sub-Sub-Sektion 4.7.2.3 näher eingegangen wird. Zuerst wird die *this.dbConnectorService.getDBs()*-Methode aufgerufen. Diese liefert ein *Observable<any>*-Objekt zurück, was bedeutet, dass der Rückgabewert erst später geliefert wird. Daher wird ein *.subscribe()* an die Methode angehängt, welches eine Funktion übergeben bekommt, welche dann aufgerufen wird, wenn *this.dbConnectorService.getDBs()* eine Antwort zurück gibt. Die Funktion, welche innerhalb der ersten *.subscribe* Methode definiert ist, übernimmt die Liste, welche *getDBs()* zurück gibt, in eine Variable namens *dataDBs*. Der lokalen *databases*-Variable wird der Inhalt der *dataDBs* direkt zugewiesen. Die Liste von Datenbank-Namen kann leer sein, daher wird im nächsten Schritt mit einer if-Abfrage überprüft ob die Länge der Liste der Datenbank-Namen nicht null ist. Sollte sie nicht leer sein, wird die lokale Variable *selectedDatabase*, welche den Namen der zurzeit ausgewählte Sub-Datenbank speichert, auf den ersten Eintrag der *this.databases* Liste gesetzt.

Danach wird die *this.dbConnectorService.getCollections()*-Methode mit dem Datenbank-Namen aus *this.selectedDatabase* als Übergabewert aufgerufen. Hier wird, wie bei beim Laden der Datenbank-Namen, *.subscribe()* verwendet. Die Liste von Collections wird, sobald die *getCollections()*-Methode die Daten erhalten hat, in die *dataCols* Variable gespeichert. Diese wird dann der *collections* Liste

zugewiesen. Zuletzt wird überprüft ob die Liste von Collection-Namen nicht leer ist. Sollte das nicht der Fall sein, wird der Variable *selectedCollection* der erste Wert aus der Liste *collections* zugewiesen. Nach dem zuweisen der *selectedCollection* Variable wird noch die Methode *updateCollectionView()* aufgerufen

Dieser Code holt somit alle Datenbank-Namen und alle Collection-Namen aus der ersten Sub-Datenbank und speichert sie in den Listen *databases* und *collections*. Außerdem werden die ersten Werte aus den beiden Listen als standardmäßig ausgewählte Werte zugewiesen.

Neben den Dropdowns ist im HTML der general-view Component auch eine Instanz der collection-view Component definiert. Das HTML hierfür sieht wie folgt aus:

```
<app-collection-view [dbName]="this.selectedDatabase"
            [collectionName]="this.selectedCollection"></app-collection-view>
```

Die collection-view Component nutzt Input-Variablen um die Variablen *this.selectedDatabase* und *this.selectedCollection* aus der general-view Component zu übernehmen. Hierfür sind in der Klasse *CollectionViewComponent* folgende Variablen definiert:

```
@Input() dbName: string = "";
@Input() collectionName: string = "";
```

Außerdem ist in der general-view Component eine Referenz-Variable zur collection-view Component namens *collectionView* hinterlegt. Diese ist wie folgt definiert:

```
@ViewChild(CollectionViewComponent) collectionView!: CollectionViewComponent;
```

Da das Übergeben der Variablen *this.selectedDatabase* und *this.selectedCollection* nur beim ersten Laden der Webseite erfolgt, gibt es in der *GeneralViewComponent*-Klasse eine Methode, welche die Werte aktualisiert. Diese Methode heißt *updateCollectionView()* und ist wie folgt definiert:

```
updateCollectionView() {
    this.collectionView.dbName = this.selectedDatabase;
    this.collectionView.collectionName = this.selectedCollection;
    this.collectionView.loadDocuments();
}
```

Hier werden die Variablen der collection-view Component neu zugewiesen und die Methode *loadDocuments()* aufgerufen, welche in der der Sub-Sub-Sektion 4.7.2.3 erläutert wird.

Alle Dropdowns sind mit einer Methode verknüpft, welche ausgeführt wird, wenn ein neuer Wert ausgewählt wird. Alle drei Dropdowns geben außerdem ein *event*-Objekt mit, welches Informationen über den neu ausgewählten Wert beinhaltet.

Das Dropdown, welches für die Auswahl der Sub-Datenbank verantwortlich ist, ruft die Methode *onDBOptionChange()* auf. Diese Methode liest den neu ausgewählten Wert aus dem *event*-Objekt aus, welches das Dropdown der Methode übergibt. Dann wir der Name der neu ausgewählten Sub-Datenbank in *selectedDatabase* gespeichert und der selbe Code wie in der *loadData()*-Methode ausgeführt, um die Variable *collections* und *selectedCollection* neu zuzuweisen. Am Ende wird *updateCollectionView()* aufgerufen. Der Code für die Methode ist wie folgt implementiert:

```
public onDBOptionChange(event: Event): void {
    this.selectedDatabase = (event.target as HTMLSelectElement).value;
    this.dbConnectorService.getCollections(this.selectedDatabase)
    .subscribe((data)=>{
        this.collections = data.content
        this.selectedCollection = this.collections[0];
        this.updateCollectionView()
    });
}
```

Durch das Aktualisieren der *this.collections* Liste werden auch die Optionen des Dropdowns, welches für die Auswahl der Collection zuständig ist, aktualisiert, da dieses direkt auf die Liste zugreift um die Optionen anzuzeigen.

Dieses Dropdown ruft die Methode *onCollOptionChange()* auf. Diese Methode weißt der Variable *selectedCollection* den neu ausgewählten Wert zu, welcher aus dem *event* Objekt ausgelesen wird. Außerdem wird die *updateCollectionView()*-Methode aufgerufen.

Das dritte Dropdown, welches den Anzeige-Typ ändert, ruft die Methode *onDisplayTypeChanged()* auf. Diese greift auf die Variable *this.collectionView* zu, welches die collection-view Component repräsentiert. Diese Component besitzt eine Variable namens *displaytype*, welcher der neu ausgewählte Wert aus dem Dropdown zugewiesen wird.

4.7.2.5 Daten anzeigen

Das Anzeigen der Dokumente erfolgt in der collection-view Component und der document-view Component. Für das Erhalten der Dokumente wird wieder der *DBConnectorService* verwendet. Die

Component nutzt Instanzen des *FilterService* und *SortingService*, welche in Sub-Sub-Sektionen 4.7.2.6 und 4.7.2.7 behandelt werden. Der Konstruktor, wo diese injiziert, werden ist in folgendem Code ersichtlich

```
constructor(private dbConnectorService: DBConnectorService, private
↪  filterService: FilterService, private sortingService: SortingService) {}
```

Die document-view Component erhält die aktuell ausgewählte Sub-Datenbank und Collection von der general-view Component. Diese werden in den Variablen *dbName* und *collectionName* gespeichert. Die zurzeit geladenen Dokumente werden in einer Liste namens *documents* gespeichert.

Das Laden der Dokumente erfolgt in der *loadDocuments()*-Methode, welche regelmäßig durch die general-view Component aufgerufen wird. Diese Methode nutzt die *getDocuments()*-Methode vom DBConnectorService. Die *getDocuments()*-Methode übernimmt die Variablen *dbName* und *collectionName* als Übergabeparameter. Da *getDocuments()* ein *Observable* Objekt zurück gibt, wird ein *.subscribe()* an die Methode angehängt, welches dann eine Funktion ausführt, sobald die Dokumente aus der Collection vom *DBConnectorService* erhalten wurden. Der Code für die Methode *loadDocuments()* ist wie folgt definiert:

```
public loadDocuments() {
    this.dbConnectorService.getDocuments(this.dbName, this.collectionName)
      .subscribe((data) => {
        this.documents = data.content;
        this.attributes = this.determineAttributes();
        this.filterOptions = [];
        this.filterKeyIndex = 0;
        this.setPossibleFilterOperators();
        this.filterValue = "";
        this.updateDocuments();
        this.selectPlaceholder();

        this.sortAttribute = "";
        this.sortAsc = true;
        this.sort("_id");
    })
}
```

Nach dem Laden der Daten wird der *documents* Liste der Inhalt der erhaltenen Dokumentenliste zugewiesen. Dann wird die *determineAttributes()*-Methode aufgerufen. Alle weiteren Zeilen sind für das Zurücksetzen der Filter- oder Sortieroptionen oder das Anwenden der Filteroptionen zuständig, welche genauer in Sub-Sub-Sektion 4.7.2.6 und 4.7.2.7 erläutert werden.

Die Methode *determineAttributes()* ist für das Anzeigen der Dokumente relevant. Diese Methode durchläuft einmal die Liste der Dokumente und fügt den Namen eines jeden Attributs zu einer Liste von Attributnamen namens *attributes* hinzu. Damit ein Attribut, welches mehrere Dokumente besitzen, nicht mehrmals gespeichert wird, wird mit einer if-Abfrage und der *.includes()*-Methode überprüft, ob das Attribut noch nicht in der Liste gespeichert ist. Diese if-Abfrage, in Kombination mit dem Durchlaufen der Dokumente, ist wie folgt definiert:

```
for (let doc of this.documents) {
      for (let k of this.getAttributes(doc)) {
        if (!ret.includes(k)) {
          ret.push(k);
        }
      }
    }
```

Nach dem Laden der Dokumente und der Attribute werden die Dokumente auf der Webseite angezeigt. Hierbei ist aber relevant, welcher Anzeige-Typ in der general-view Component ausgewählt ist. Der Anzeige-Typ wird in der Variable *displaytype* gespeichert und kann die Werte *List* und *Blobs* annehmen. Je nachdem welchen Wert die Variable speichert wird ein anderer Teil des HTML angezeigt. Hierfür wird **ngIf* und die *isBlobs()*-Methode verwendet, welche True zurück gibt wenn *displaytype* den Wert *Blobs* beinhaltet, sonst wird False zurückgegeben. Wenn der Anzeige-Typ *Blobs* ist, wird folgender Teil des HTML angezeigt:

```
<div class=container *ngIf="isBlobs()">
    <label *ngFor="let doc of this.getDocuments()">
        <app-document-view [document]="doc"></app-document-view>
    </label>
</div>
```

Hier wird **ngFor* verwendet um für jedes Dokument eine document-view Component anzuzeigen. In Abbildung 27 ist zu sehen wie die Dokumente auf der Webseite unter Verwendung des *Blobs* Anzeige-Typs angezeigt werden.

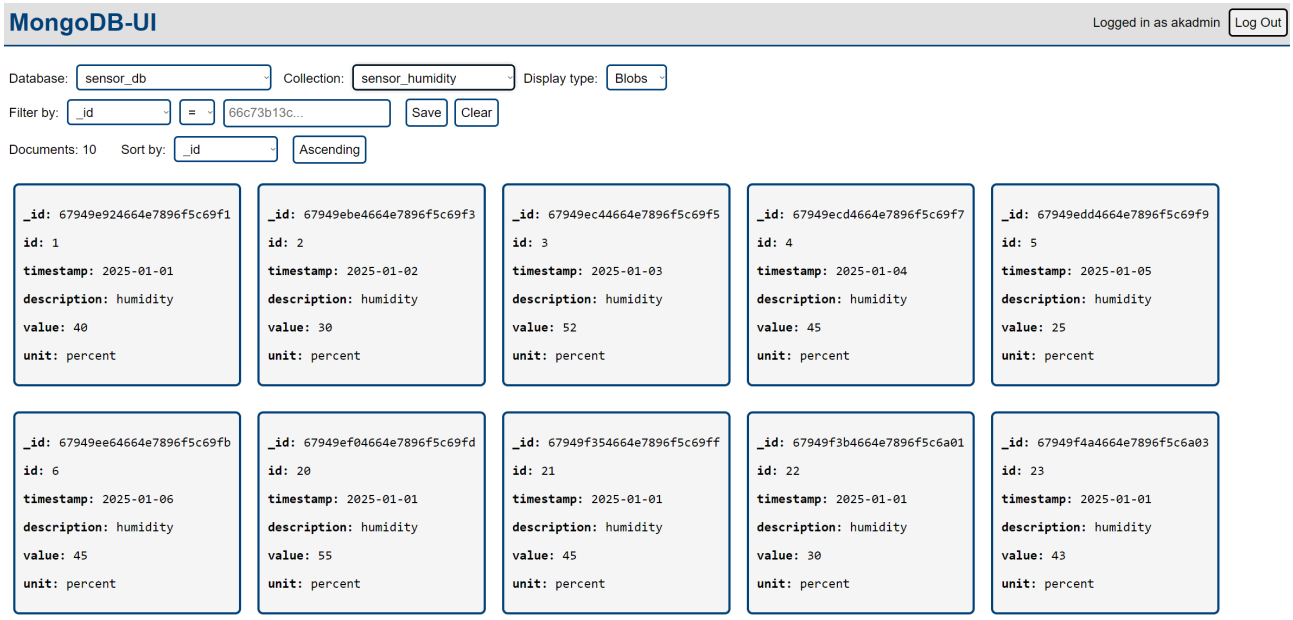


Abbildung 27: MongoDB-UI unter Verwendung des Blobs Anzeige-Typs

Ein einzelnes Dokument wird durch eine document-view Component angezeigt. Hierfür wird zuerst das Dokument Objekt an die document-view Component übergeben. Das erfolgt an folgender Stelle im HTML:

```
<app-document-view [document]="doc"></app-document-view>
```

Das Dokument wird dann in der document-view Component in der Variable *document* gespeichert. Hierfür wird der *@Input* Decorator verwendet, welcher ermöglicht, dass Variablen über HTML übergeben werden. Die Variable ist wie folgt definiert:

```
@Input() document: any = {};
```

Das Anzeigen des Dokuments erfolgt im HTML des document-view Component, welches wie folgt definiert ist:

```
<div class="documentView">
    <label *ngFor="let k of this.getAttributes()">
        <p>
            <label class="key">{{k}}: </label>
            <label *ngIf="!isObject(document[k])">{{document[k]}}</label>
            <label *ngIf="isObject(document[k])"
             ↪  [innerHTML]="getObjectString(document[k])"></label>
        </p>
```

```
    </label>
</div>
```

Hier wird mittels **ngFor* für jedes Attribut des Dokuments eine Zeile geschrieben, welche das Attribut, auch *key* genannt, und den Wert des Attributs beinhaltet. Sollte das Attribut ein Objekt als Wert haben, wird mit der *getObjectString()*-Methode ein JSON-String aus dem Objekt generiert, welche mit Zeilenumbrüchen und Tabstops erweitert wird um die Lesbarkeit des Objekts zu verbessern. Ein Dokument mit einem Objekt Attribut ist in Abbildung 28 zu sehen.

Abbildung 28: Dokument in Blob-Ansicht mit Objekt Attribut

Sollte als Anzeige-Typ *List* ausgewählt sein, werden die Dokumente als Zeilen in einer Tabelle generiert. Die Spalten der Tabelle werden außerdem durch die unterschiedlichen Dokument-Attribute in der Collection definiert. Für das Erzeugen der Tabelle ist folgendes HTML in der collection-view Component verantwortlich:

```
<table>
    <tbody>
        <tr>
            <th *ngFor="let k of attributes" (click)="sort(k)">
                {{k}}
                <span *ngIf="this.isSorting(k,false)"> &#9650;</span>
                <span *ngIf="this.isSorting(k,true)"> &#9660;</span>
                <span *ngIf="k!=this.sortAttribute">
                       
                </span>
            </th>
        </tr>
```

```
        <tr *ngFor="let doc of this.getDocuments()">
            <td *ngFor="let k of attributes">
                <div *ngIf="!isObject(doc[k])">{{doc[k]}}</div>
                <div *ngIf="isObject(doc[k])">
                     Object display not supported in list mode
                </div>
            </td>
        </tr>
    </tbody>
</table>
```

Für das Erzeugen der Kopfzeile der Tabelle wird **ngFor* verwendet. Die Kopfzeile wird mit den Attributen aus der *attributes* Liste befüllt.

Für das Auflisten der Dokumente als Zeilen in der Tabelle wird **ngFor* verwendet, welches auf die Methode *this.getDocuments()* zugreift, um die Dokument-Liste zu erhalten. Es wird dann eine zweite **ngFor*-Definition verwendet, um die Attribut-Werte in der Zeile aufzulisten. Hierfür wird wieder auf die *attributes* Liste zugegriffen. Es kann vorkommen, dass ein Dokument für ein Attribut keinen Wert gespeichert hat. Beim direkten Zugriff auf den Attribut-Wert eines Dokuments mittels *doc[k]* wird dann ein leeres Feld angezeigt. Sollte ein Attribut ein Objekt als Wert gespeichert haben, wird die Meldung *Öbject display not supported in list mode"* als Wert angezeigt. Das ist der Fall, da das Anzeigen eines großen Objektes in einer Tabellen-Zelle die Spalten stark verzerren könnte. Ein Beispiel der Tabelle auf der Webseite ist in Abbildung 29 zu sehen.

_id ▼	name	age	city	company	job	address
675088fc3a0e8eefa71fae59	Klaus	18	Perg			
6750893d51a796f2587813a5	John	20	Linz	Fabasoft	Software Entwickler	
67c0aecc49c37a91d22b313e	Ben	25	New York			Object display not supported in list mode

Abbildung 29: Dokumente in List Ansicht

4.7.2.6 Filtern

Die Filter-Funktionalität ist verteilt in der collection-view Component und der *FilterService*-Klasse implementiert.

Um die Dokumente zu filtern müssen drei Variablen in der collection-view Component bestimmt werden. Diese Variablen sind folgende:

filterKeyIndex: *filterKeyIndex* beinhaltet den Index eines Attributs in der *attributes* Liste. Das Attribut, auf welches der Index zeigt, wird dann für das Filtern verwendet.

filterValue: *filterValue* beinhaltet den Wert, mit dem das durch *filterKeyIndex* ausgewählte Attribut der Dokumente verglichen wird.

filterOperator: *filterOperator* bestimmt, wie die Attribut-Werte des Dokuments mit dem *filterValue* verglichen werden.

Die Werte dieser Attribute werden durch ein Filter-Menü bestimmt, welches in der collection-view Component definiert ist.

Filter-Auswahl

filterKeyIndex wird durch ein Dropdown bestimmt, welches im HTML der collection-view Component wie folgt definiert ist:

```
<select [(ngModel)]="this.filterKeyIndex" class="FilterKeyDropdown"
↪  id="filterKeyDropdown"
        (change)="updateFilterKey($event)">
        <option *ngFor="let option of this.attributes; let i = index"
        [value]="i" class="filterKeyDropdownOption">
            {{option }}</option>
    </select>
```

Das Dropdown stellt alle unterschiedlichen Attribute in der Collection als Auswahloptionen zur Verfügung. Das wird durch das Nutzen von **ngFor* mit Zugriff auf die *attributes* Liste möglich. Durch die *ngModel* Verknüpfung mit der *filterKeyIndex* Variable wird der Wert der Variable beim Verwenden des Dropdowns auf den Index des ausgewählten Werts, innerhalb der *attributes* Liste, gesetzt. Das Verwenden des Dropdowns ruft außerdem die *updateFilterKey()*-Methode auf.

Die Methode *updateFilterKey()* setzt die Variablen *filterValue* und *filterOperator* auf Standardwerte zurück. Weiters werden die Methoden *this.updateDocuments()*, *this.selectPlaceholder()* und *this.setPossibleFilterOperators()* aufgerufen.

Die *this.selectPlaceholder()*-Methode ermittelt den Datentyp, welches das ausgewählte Attribut in den Dokumenten hat und speichert in der *filterTextPlaceholder* Variable einen passenden Platzhalter-Text, welcher dann im Eingabefeld für die *filterValue* Variable angezeigt wird. Die *this.setPossibleFilterOperators()*-Methode ermittelt auch den Datentyp des Attributs und speichert dann die erlaubten Filter-

Operationen in die Liste *visibleOperators*. Der Filter Operator -ïst für alle Datentypen freigeschaltet, die Filter Operatoren «ünd » sind nur für Attribute mit den Datentypen *Number* und *Date* freigeschaltet.

Der Wert der *filterOperator* Variable wird mit einem Dropdown festgelegt, welches durch folgendes HTML der collection-view Component definiert ist:

```
<select class="FilterOperatorDropdown" id="filterOperatorDropdown"
[(ngModel)]="this.filterOperator" (change)="updateDocuments()">
    <option *ngFor="let option of this.visibleOperators" [value]="option">
    {{ option }}</option>
</select>
```

Die Auswahlmöglichkeiten werden aus der *visibleOperators* Liste bezogen. Der ausgewählte Wert wird durch eine *ngModel*-Verknüpfung in die Variable *filterOperator* gespeichert.

Das Dropdown kann folgende Filter Operatoren zur Verfügung stellen:

=: Dieser Operator überprüft, ob der Wert, welchen das ausgewählte Attribut in einem Dokument speichert, den Wert, der in der *filterValue* Variable steht **beinhaltet**. Sollte ein Dokument beim ausgewählten Attribut den Wert "Programmieren"haben, und die *filterValue* Variable den Wert "gram"beinhalten, schafft es das Dokument durch den Filter.

>: Dieser Operator überprüft, ob der Wert, welchen das ausgewählte Attribut in einem Dokument speichert, größer ist als der Wert, welcher in der *filterValue* Variable steht. Dieser Operator gilt für Zahlen und für Daten. Im Falle eines Datums wird "größer alsäls ßpäter als"verstanden. Somit ist der 8. Juni 2025 "größer als"der 5. Mai 2014.

<: Dieser Operator überprüft, ob der Wert, welchen das ausgewählte Attribut in einem Dokument speichert, kleiner ist als der Wert, welcher in der *filterValue* Variable steht. Es gelten die gleichen Anwendungsfälle wie beim »Öperator.

Der Wert der *filterValue* Variable wird durch ein Eingabefeld bestimmt, welches im HTML der collection-view Component wie folgt definiert ist:

```
<input [(ngModel)]="this.filterValue" type="text" class="filter-input"
    [ngClass]="getInputClass()" (input)="updateFilterValue($event)"
    (keydown.enter)="saveFilter()" [placeholder]="filterTextPlaceholder"
    (focus)="showInputTooltip = true" (blur)="showInputTooltip = false">
```

Das Eingabefeld schreibt durch *[(ngModel)]* den eingegebenen Text in die Variable *filterValue*. Mittels *[ngClass]* wird die CSS-Klasse des Eingabefelds durch die Methode *getInputClass()* definiert. Dadurch ist es möglich das Eingabe-Feld rot zu färben, sollte ein invalides Datum oder einen String für ein Nummer Attribut eingeben werden, wie in Abbildung 30 zu sehen ist.

Abbildung 30: Invalide Filter Eingabe

Durch *(input)* wird festgelegt, dass bei jeglicher Änderung im Eingabefeld die Methode *updateFilterValue()* aufgerufen wird. Diese ruft dann die Methode *updateDocuments()* auf. Mittels *[Placeholder]* wird der Platzhalter definiert, welcher in *filterTextPlaceholder* gespeichert wird. Mit *(keydown.enter)* wird festgelegt, dass die *saveFilter()*-Methode aufgerufen wird, wenn der Enter-Knopf gedrückt wird während das Eingabefeld ausgewählt ist. Die *(focus)* und *(blur)* Konfigurationen setzen die Variable *showInputTooltip* auf True und False.

Die *showInputTooltip* Variable bestimmt, ob ein eine Info-Box unterhalb des Eingabefelds angezeigt wird, welche den Nutzer darauf hinweist wie Zahlen- und Datums-Eingaben formatiert sein sollten. Die Info-Box ist in Abbildung 31 zu sehen.

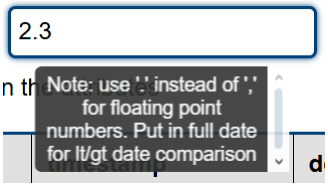


Abbildung 31: Info-Box unterhalb des Eingabe Felds

Das Filtersystem ermöglicht es, mehrere Filter gleichzeitig zu speichern. Hierfür werden alle aktiven Filter in einer Liste namens *filterOptions* gespeichert. Ein Filter wird als ein Objekt mit folgendem Format gespeichert:

```
FilterOption {
    filterKey: string;
    filterValue: string;
    filterOperator: string;
}
```

Das *filterKey* Attribut speichert im Gegensatz zur *filterKeyIndex* Variable direkt den Namen des Attributs, nach dem gefiltert werden soll. Der zurzeit aktive Filter wird in der Liste gespeichert, wenn die Methode *saveFilter()* aufgerufen wird. Diese Methode wird durch das Eingabefeld und durch einen Save-Button, welcher sich neben dem Eingabefeld befindet, aufgerufen. Die *saveFilter()*-Methode

erstellt aus dem aktuell ausgewählten Filter ein Filter Objekt und speichert es in der Liste. Der Code für dieses Vorgehen ist wie folgt definiert:

```
let fo: FilterOption = { filterKey: this.attributes[this.filterKeyIndex],
 ↪  filterValue: this.filterValue, filterOperator: this.filterOperator };
this.filterOptions.push(fo);
```

Die gespeicherten Filteroptionen werden neben den Eingabefeldern für das Filtersystem aufgelistet. Die Filteroptionen können einzeln durch Anklicken gelöscht werden. Es ist außerdem ein Clear-Button vorhanden, welcher alle gespeicherten Filter löscht. Die Auflistung der Filteroptionen, sowie der Clear-Button sind in Abbildung 32 zu sehen.

Abbildung 32: Filter Menü mit gespeicherten Filtern

Filter Anwenden

Die Filter werden jedes Mal auf die Dokumente in der *documents* Liste angewendet, wenn die Methode *updateDocuments()* aufgerufen wird. Die *updateDocuments()*-Methode erstellt eine Kopie der aktuellen *filterOptions* Liste und fügt die aktuellen Filter-Einstellungen als neues *FilterOption* Objekt zur Liste hinzu. Für das Anwenden der Filteroptionen ist die Klasse *FilterService* zuständig. Diese besitzt eine Methode namens *filterDocs()*, welche die Dokumente und die Filter-Optionen als Übergabeparameter übernimmt und eine Liste von Dokumenten zurück gibt, welche gefiltert wurden. In der *updateDocuments()*-Methode werden die gefilterten Dokumente in der Liste *filteredDocuments* gespeichert. Die gesamte Methode *updateDocuments()* ist wie folgt definiert:

```
public updateDocuments() {
    let filter: FilterOption = { filterKey:
     ↪  this.attributes[this.filterKeyIndex], filterValue: this.filterValue,
     ↪  filterOperator: this.filterOperator };
    let filterOptionsCopy: FilterOption[] = [filter];
    for (let fo of this.filterOptions) filterOptionsCopy.push(fo);
    this.filteredDocuments = this.filterService.filterDocs(this.documents,
     ↪  filterOptionsCopy);
  }
```

Auf der Webseite wird die Liste *filteredDocuments* angezeigt, wenn die aktuellen Filter-Einstellungen nicht leer sind oder die Liste *filterOptions* nicht leer ist. Diese Logik ist in der Methode *getDocuments()* implementiert, welche die collection-view Component verwendet um die Dokumente aufzulisten.

Die *filterDocs()*-Methode in der *FilterService*-Klasse durchläuft die Liste von Filter-Optionen und wendet dann jeden Filter einzeln auf alle Dokumente in der übergebenen Dokumenten Liste an. Das Anwenden eines Filters erfolgt in der *applyFilter()*-Methode, welche eine Liste von Dokumenten und eine Filter-Option als Übergabeparameter übernimmt.

In der *applyFilter()*-Methode wird dann jedes Dokument durchlaufen. Zuerst wird überprüft, ob ein Dokument überhaupt das Attribut, welches im übergebenen *FilterOption* Objekt definiert ist, besitzt. Dann wird aus dem *FilterOption* Objekt ausgelesen, welche Filter-Operation auf das Dokument angewandt werden soll. Je nach Filter-Operation wird eine der folgenden Methoden aufgerufen:

- Filter-Operation $=$: docContainsFilterValue()
- Filter-Operation $<$: docLtFilterValue()
- Filter-Operation $>$: docGtFilterValue()

Alle drei übernehmen als Übergabeparameter das Dokument, welches überprüft werden soll, das Filter-Attribut und den Filter-Wert, wobei die letzten beiden im übergebenen *FilterOption* Objekt gespeichert sind. Die Methoden geben dann *True* zurück wenn das Dokument dem Filter entspricht und in der Liste bleiben soll und False, wenn es dem Filter nicht entspricht und nicht angezeigt werden soll. Da *applyFilter()* eine Liste von Dokumenten zurück gibt, welche dem übergebenen Filter entsprechen, wird das Dokument, wenn die Methode True zurück gibt, in eine Liste gespeichert, welche am Ende der Methode zurückgegeben wird. Die gesamte *applyFilter()*-Methode ist wie folgt definiert:

```
public applyFilter(docs: any[], filter: FilterOption){
    if(filter.filterValue == "") return docs;

    let filteredDocs: any[] = []
    for (let doc of docs) {
      if (this.getAttributes(doc).includes(filter.filterKey)) {
        switch (filter.filterOperator) {
          case "=":
            if (this.docContainsFilterValue(doc, filter.filterKey,
            ↪  filter.filterValue)) filteredDocs.push(doc);
            break;
```

```
          case "<":
            if (this.docLtFilterValue(doc, filter.filterKey,
            ↪  filter.filterValue)) filteredDocs.push(doc);
            break;

          case ">":
            if (this.docGtFilterValue(doc, filter.filterKey,
            ↪  filter.filterValue)) filteredDocs.push(doc);
            break;
        }
      }
    }
    return filteredDocs;
}
```

4.7.2.7 Sortieren

Das Sortieren der Dokumente erfolgt in der collection-view Component mit Unterstützung der *SortingService*-Klasse. Wie die Dokumente sortiert werden, hängt von zwei Variablen ab:

sortAttribute: Diese Variable enthält den Namen des Attributs, nach dem die Dokumente sortiert werden soll in Form eines Strings.

sortAsc: Diese Variable enthält einen Boolean. Wenn der Boolean *True* ist, werden die Dokumente aufsteigend sortiert, wenn er *False* ist werden sie absteigend sortiert.

Die Variable *sortAttribute* speichert standardmäßig den Wert "_idünd die Variable *sortAsc* ist standardmäßig auf *True* gesetzt. Die Art, wie der Benutzer die Dokumente sortieren kann, hängt davon ab, welcher Anzeige-Typ gerade ausgewählt ist.

Sortieroptionen Auswählen

Wenn als Anzeige-Typ *Blobs* ausgewählt ist, wird ein Sortier-Menü angezeigt, welches ein Dropdown besitzt, in dem man das Attribut, nach dem Sortiert werden soll, auswählen kann. Das Menü besitzt auch einen Button, welcher die Sortier-Richtung wechseln kann. Das Menü ist aufgrund einer **ngIf* Definition nur sichtbar wenn der Anzeige-Typ *Blobs ist*. Das Sortier-Menü ist in Abbildung 33 zu sehen

Abbildung 33: Das Sortier-Menü für den Blobs Anzeige-Typ

Das Dropdown bezieht die Optionen aus der *attributes* Liste. Wenn im Dropdown ein neuer Wert ausgewählt wird, wird die Methode *blobViewSort()* aufgerufen. Diese Methode liest das neu ausgewählte Attribut aus der *event* Variable aus, welche das Dropdown übergibt und ruft mit der Variable die *sort()*-Methode auf, die in der Component definiert ist.

Der Button ruft die Methode *swapSortingDirection()* auf. Außerdem wird der Text im Button angepasst auf die aktuelle Sortier-Richtung. Das ist durch *Conditional Text Rendering* möglich, welches einen anderen Text anzeigt, wenn die Variable *sortAsc* einen anderen Wert hat. Im HTML ist der Buttom mit *Conditional Text Rendering* wie folgt definiert:

```
<button (click)="swapSortingDirection()">
    {{this.sortAsc ? 'Ascending': 'Descending'}}
</button>
```

Die Methode *swapSortingDirection()* ruft direkt die Methode *sort()* mit dem aktuellem Sortier-Attribut auf. Die *sort()* dreht die Sortier-Richtung um, wenn sie mit dem selben Sortier-Attribut aufgerufen wird, welches bereits in der *sortAttribute* Variable gespeichert ist. Das sorgt dafür, dass durch den Methoden-Aufruf in der *swapSortingDirection()*-Methode die Sortier-Richtung umgedreht wird. Diese Logik der *sort()*-Methode wird durch folgenden Code ermöglicht:

```
if (attribute == this.sortAttribute) {
      this.sortAsc = !this.sortAsc;
}
```

Im Anzeige-Typ *List* erfolgt das Sortieren durch das Klicken auf die Zellen in der Kopfzeile der Tabelle. Diese speichern die Attribute der Dokumente und rufen die *sort()*-Methode auf wenn sie angeklickt und übergeben das Attribut, welches die Zelle speichert. Wenn die *sort()*-Methode ein Attribut übergeben bekommt und dieses nicht dasselbe ist, wie jenes, welches in der *sortAttribute* Variable gespeichert wird, wird es mit dem neuen Attribut überschrieben. Diese Logik sorgt dafür, dass beim ersten Klicken auf eine Kopfzeilen-Zelle, die *sortAttribute* Variable den neuen Attribut-Wert speichert, und beim erneuten Klicken, die Sortier-Richtung umgedreht wird. Die gesamte Logik wird durch folgenden Code in der *sort()*-Methode ermöglicht:

```
if (attribute == this.sortAttribute) {
    this.sortAsc = !this.sortAsc;
} else {
    this.sortAttribute = attribute;
    this.sortAsc = true;
}
```

Sortierung durchführen

Am Ende der *sort()*-Methode werden die beiden Listen *documents* und *filteredDocuments* sortiert. Das erfolgt, indem die *Array.sort()*-Methode der beiden Listen aufgerufen wird. Die *Array.sort()*-Methode erwartet, dass ihr eine Funktion übergeben wird, welche zwei Variablen als Übergabeparameter akzeptiert und eine Zahl zurück gibt. Diese Funktion wird benötigt, da die *Array.sort()*-Methode eine Vergleichsbasis für die Dokumente benötigt. In der *sort()*-Methode wird die *Array.sort()* wie folgt angewandt:

```
this.documents.sort((a, b) =>
this.sortingService.sortFunction(a[attribute], b[attribute], this.sortAsc));
this.filteredDocuments.sort((a, b) =>
this.sortingService.sortFunction(a[attribute], b[attribute], this.sortAsc));
```

Die Variablen *a* und *b* repräsentieren zwei Dokumente, welche die *Array.sort()*-Methode vergleicht. Damit das Sortieren funktioniert, muss die Funktion, die der *Array.sort()*-Methode übergeben wird, eine positive Zahl zurückgeben, wenn das Dokument in der Variable *a* in der sortierten Liste vor dem Dokument in der Variable *b* platziert werden soll.

Der eigentliche Vergleich der Dokumente findet in der *sortFunction()*-Methode der *SortingService*-Klasse statt. Der Methode werden die Werte des Attributs, nach dem sortiert wird, der Dokumente übergeben. Außerdem wird die Sortier-Richtung übergeben. Diese Methode gibt dann eine positive Zahl zurück, wenn Dokument a vor Dokument b platziert werden soll und eine negative Zahl, wenn die Reihenfolge invertiert sein soll.

Die *sortFunction()*-Methode überprüft immer den Datentyp der Werte, die ihr übergeben werden, und verwendet dann einen passenden Vergleich. Die Methode unterstützt das Sortieren von String-, Number- und Date-Variablen. Ein Beispiel von sortierten Dokumenten mit Anzeige-Typ *List* ist in Abbildung 34 zu sehen.

_id	id	timestamp	description	value ▼	unit
67949edd4664e7896f5c69f9	5	2025-01-05	humidity	25	percent
67949f3b4664e7896f5c6a01	22	2025-01-01	humidity	30	percent
67949ebe4664e7896f5c69f3	2	2025-01-02	humidity	30	percent
67c31075bf21be107f6141c0	11	2025-02-28T09:00:00	humidity	38	percent
67949e924664e7896f5c69f1	1	2025-01-01	humidity	40	percent
67949f4a4664e7896f5c6a03	23	2025-01-01	humidity	43	percent
67c3107fbf21be107f6141c3	12	2025-02-28T10:00:00	humidity	43	percent
67c30facbf21be107f6141b4	7	2025-03-01T09:00:00	humidity	43	percent
67c30fb1bf21be107f6141b7	8	2025-03-01T09:00:00	humidity	43	percent

Abbildung 34: Sortierte Dokument in der Listen Ansicht

4.7.2.8 Deployment

Das Deployment des MongoDB-UI-Frontends erfolgt nach dem selben Prinzip, wie es in der Sektion 4.1 erklärt wird. Das Dockerfile beinhaltet als zusätzliche Umgebungsvariable die URL des Backend-Services.

4.7.2.9 Tests

Für die Angular-Webseite sind Unit-Tests implementiert. Diese sind für jede Component unabhängig in einer .spec.ts-Datei implementiert. Für das Implementieren und Durchführen der Tests werden die Bibliotheken *Jasmine* und *Karma-Test-Runner-Tool* verwendet, welche automatisch im Angular-Projekt installiert sind. Für jede Component werden standardmäßig zwei Tests generiert, welche überprüfen ob die Component generiert werden kann.

Weitere Tests existieren für die document-view Component und collection-view Component.

Für die document-view Component existiert ein zusätzlicher Test, welcher überprüft, ob das Umwandeln eines Objekts zu einem String erfolgreich durchgeführt wird.

Für die collection-view Component existieren mehrere Tests, welche folgende Funktionalitäten testen:

- Ermitteln der einzelnen Attribute
- Ermitteln der Attribute eines Objekts
- Überprüfen ob eine Variable ein Objekt ist.
- Überprüfen ob eine Variable eine Zahl ist.
- Filter speichern und löschen.

- Invalide Filter-Eingabe ermitteln

Ein Unit-Test wird beispielsweise wie folgt implementiert:

```
it("invalidInput() works correctly", ()=>{
    component.filterOperator = "<";
    component.filterValue = "Hello World";
    expect(component.invalidInput()).toBeTrue();
    component.filterValue = "5";
    expect(component.invalidInput()).toBeFalse();
});
```

Dieser Unit-Test überprüft ob eine Filter-Eingabe richtig als ßulässigöder "nicht zulässigërkannt wird. Hierfür werden in der Component, welche für den test temporär generiert wird, die Variablen *filterOperator* und *filterValue* auf Test-Werte gesetzt. Dabei wird einmal ein Wert verwendet, welcher zulässig ist und einer, der nicht zulässig ist. Mittels *expect()* wird dann überprüft, ob die Methode *invalidInput()* die Kombination von Variablen richtig als invalide oder valide erkennt.

5 Ergebnis

In den nachfolgenden Abschnitten wird gezeigt, welche Funktionalitäten das Endergebnis des Projekts erfüllt. Außerdem werden diese mit der Zielsetzung, welche in Form von OneDev-Issues definiert sind, verglichen. Die OneDev-Issues wurden ursprünglich in Englisch verfasst, in diesem Kapitel werden die Issues aber auf Deutsch übersetzt aufgelistet. Die originalen Issues sind in der Sektion B zu sehen.

5.1 Authentik

Mithilfe von Authentik können Benutzer erstellt, Berechtigungen verteilt und Anwendungen authentifiziert werden. Dadurch wird die gesamte Authentifizierung des Systems erleichtert und Änderungen der Berechtigungen können zentral für jeden Service verwaltet und verändert werden. Das Dashboard bietet zusätzlich die Möglichkeit, alle Dienste des Systems aufzurufen und auf einen Blick zu sehen.

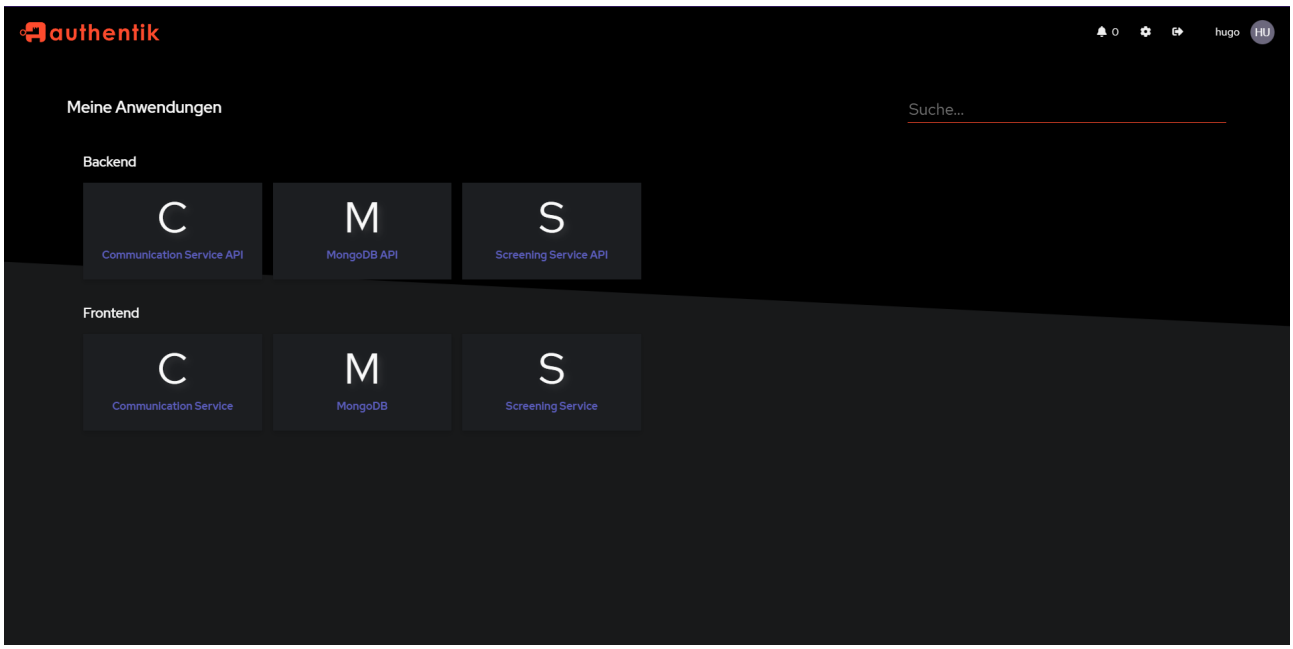


Abbildung 35: Authentik Dashborad

5.1.1 Vergleich mit OneDev Issues

- Issue #4: Authentik Aufsetzen: Authentik soll aufgesetzt werden, um die Authentifizierung der Applikationen durchzuführen.
- Issue #6: Rollen für Authentik: In Authentik sollen zwei Rollen konfiguriert werden. Eine Adminrolle, mit der der Administrator einstellen kann, wer auf welche Applikationen zugreifen kann. Mit der Userrolle soll man sich einloggen können und auf Applikationen zugreifen können.

Alle der oben genannten Issues wurden erfolgreich implementiert.

5.2 Module

5.2.1 Authentication-Module

Es wurde ein Modul entwickelt, das Funktionen zur Authentifizierung bereitstellt, um den Zugriff auf API-Endpunkte der verschiedenen Python-Backends abzusichern. Dabei kommt Authentik in Kombination mit OpenID Connect zum Einsatz. Das Modul stellt sicher, dass nur authentifizierte Benutzer Zugriff auf die jeweiligen Services erhalten.

5.2.1.1 Vergleich mit den OneDev-Issues

- Issue #15: Authentifizierung für Screening und Communication Service Backend: Die Backend-APIs des Screening- und Communication-Service sollen OpenID Connect und Authentik verwenden, um Benutzer zu autorisieren. Kein Inhalt soll ohne Login zugänglich sein.
- Issue #25: Authentifizierung für MongoDB Backend: Das MongoDB-Backend soll im Authentik-Deployment für nicht eingeloggte Benutzer nicht zugänglich sein.
- Issue #26: Authentifizierung für Screening Service Backend: Die Screening-Service-Backend-API soll im Authentik-Deployment für nicht eingeloggte Benutzer nicht zugänglich sein.

5.2.2 Secret-Communication-Module

Es wurde ein Modul entwickelt, das die Interaktion mit Kubernetes Secrets ermöglicht. Ziel war es, die Konfiguration von Endpunkten dynamisch zu gestalten. Durch die Manipulation der Secrets können Informationen gleichzeitig an mehrere Output-URLs gesendet werden.

5.2.2.1 Vergleich mit den OneDev-Issues

- Issue #16: Setzen von Input- und Output-URLs: Der Communication-Service soll um Endpunkte erweitert werden, die es dem UI ermöglichen, Output-URLs zu setzen. Es soll möglich sein, mehrere URLs zu speichern, abzufragen und zu entfernen.

5.2.3 Database-Module

Es wurde ein Modul entwickelt, welches Funktionen enthält, um Daten auf mehreren separaten MongoDB-Datenbanken zu speichern. Zudem wurde ein API-Router implementiert, welcher über folgende Endpoints verfügt:

- Alle aktuellen MongoDB-URLs abrufen
- Eine einzige MongoDB gefiltert nach Namen abrufen
- Eine neue MongoDB-URL hinzufügen
- Den Status einer MongoDB-URL verändern
- Eine MongoDB-URL löschen

Die Router-Endpoints des Database-Modules werden bereits im Communication-Service-Backend eingebunden und werden in der Benutzeroberflächen genutzt, um die URLs anzupassen. Die Funktion zum Speichern auf mehreren MongoDB-Instanzen wird beim Communication-Service genutzt, um neu eingehende Sensordaten auf mehreren MongoDB-Instanzen persistieren zu können. Die Funktionalitäten werden allerdings noch nicht aktiv beim Screening-Service eingesetzt, da es Probleme mit der Authentizierung beim Communication-Service-Backend gibt, welche aus zeitlichen Gründen nicht mehr gelöst werden konnten. Auf einem separaten Git-Branch des Screening-Services sind die Funktionalitäten des Database-Modules bereits eingebunden, werden allerdings erst in den Main-Branch gemerged, sobald der Fehler behoben wurde.

5.2.3.1 Vergleich mit dem OneDev-Issue

- Issue #14 Einstellen der Datenbank URL: Der Benutzer soll die Möglichkeit haben die URL der Datenbank im Frontend einzustellen.

Das Issue wurde erfolgreich implementiert.

5.3 Screening-Service

5.3.1 Frontend

Mithilfe des Frontends des Screeningservices können die Berechnungsmethoden des Screening-Service-Backend ein- und ausgeschalten werden. Es kann das Intervall der Berechnungen mittels einer CronJob-Shedule-Expression eingestellt werden. Zusätzlich besteht die Möglichkeit, die Datenbanken, in denen die Daten gespeichert werden, zu verändern, zu deaktivieren, zu aktivieren und zu löschen. Die Authentifizierung erfolgt durch den Identity-Provider Authentik.

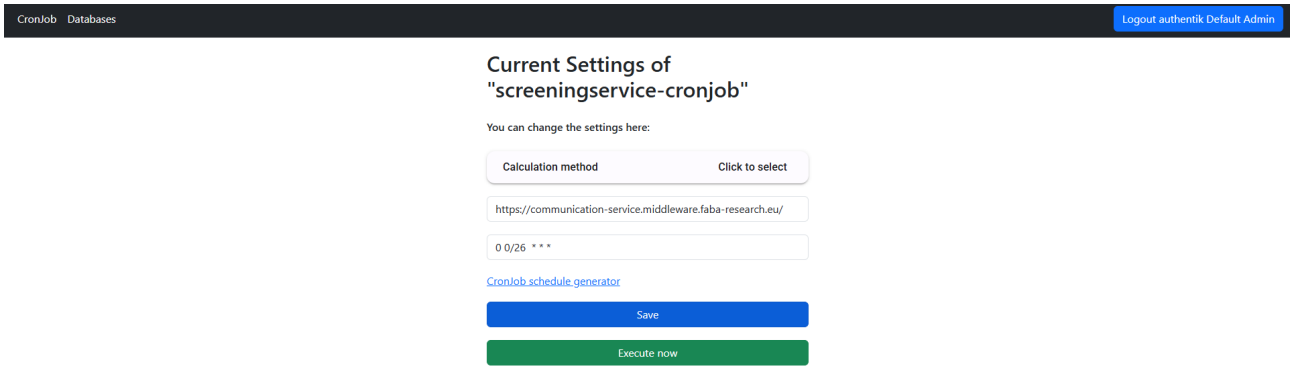


Abbildung 36: Screening Service

5.3.1.1 Vergleich der OneDev Issues

- Issue #7 Authentifizierung des Frontends: Das Frontend soll durch den Identity Provider authentifiziert werden.
- Issue #9 UI für den Screeningservice: Eine einfache grafische Benutzeroberfläche soll für den Screeningservice erstellt werden. Es soll möglich sein, dem Intervall des CronJobs auszuwählen und es soll möglich sein, den CronJob sofort auszuführen.
- Issue #13 Auswahl der Berechnunsgmethoden: Dem Benutzer soll es ermöglicht werden, die Berechnungsmethoden auszuwählen.
- Issue #14 Einstellen der Datenbank URL: Der Benutzer soll die Möglichkeit haben, die URL der Datenbank im Frontend einzustellen.

Alle der oben genannten Issues wurden erfolgreich implementiert.

5.3.2 Backend

Es wurde ein Backend für den Screening-Service implementiert, um den Screening-Service-Cronjob verwalten zu können. Dabei wurden folgende Endpoints konkret implementiert:

- Cronjob starten
- Cronjob-Intervall aktualisieren
- Vorhandene Berechnungsmethoden ein- oder ausschalten
- Communication-Service-URL bearbeiten
- Informationen kombiniert abrufen mit einer Abfrage: Communication-Service-URL, aktuelles Cronjob-Intervall, Cronjobs in einem Namespace, Berechnungsmethoden inklusive Status

5.3.2.1 Vergleich mit den OneDev Issues

- Issue #10 Neue Berechnungsmethode erstellen: Es soll eine neue Berechnungsmethode für neue Sensordaten erstellt werden. Die Sensordaten sollen an den Communication-Service gesendet werden, und dieser leitet die Information über die Berechnungen an Fabasoft Approve VDE weiter. Die Berechnungen sollen in der MongoDB gespeichert werden.
- Issue #13 Berechnungsmethoden ein- oder ausschalten: Über einen REST-Endpoint soll es möglich sein, Berechnungsmethoden des Screening-Service-Cronjobs ein- oder auszuschalten
- Issue #17 Screening-Service-Cronjob-Intervall bearbeiten: Über einen REST-Endpoint soll die Bearbeitung des Cronjob-Intervalls ermöglicht werden
- Issue #18 Berechnungsmethode setzen: Es soll ein Endpoint hinzugefügt werden um eine Berechnungsmethode zu setzen.
- Issue #22 Tests für neue Screening-Service-Erweiterungen schreiben und alte Tests erweitern

Anmerkung zu Issue #18: Es wurde kein Endpoint hinzugefügt um eine Berechnungsmethode zu setzen. Die Definition der Berechnungsmethode erfolgt stattdessen in einem Python-Objekt im Screening-Service-Cronjob. Alle anderen Issues wurden erfolgreich implementiert.

5.4 Communication-Service

5.4.1 Frontend

Es wurde ein Frontend für den Communication-Serive implementiert, um die Weiterleitungen und um die Datenbanken des Communication-Services zu konfigurieren. So können diese erstellt, aktiviert, deaktiviert und gelöscht werden. Authentifiziert wird das Frontend genauso wie das Frontend des Screening-Serivces mit dem globalen Identity-Provider Authentik.

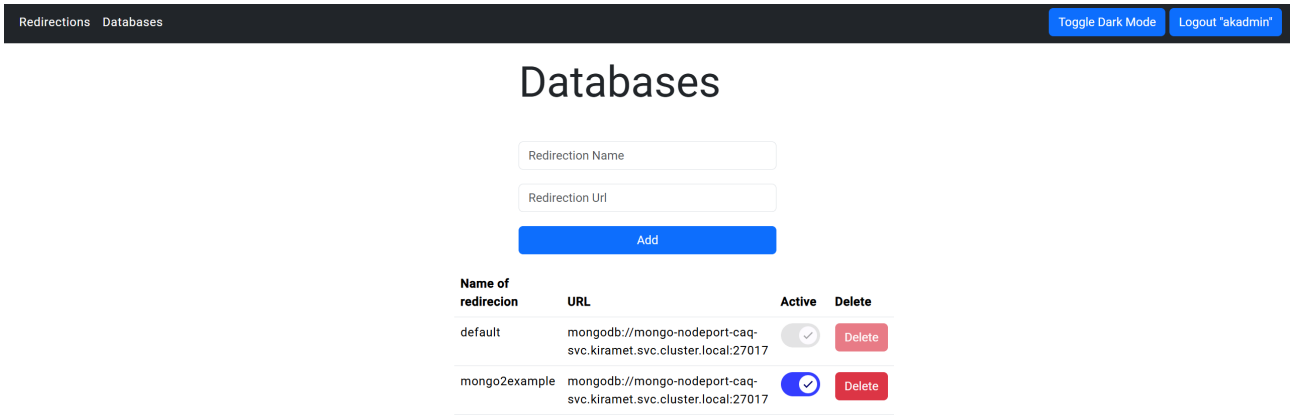


Abbildung 37: Communication Service

- Issue #7 Authentifizierung des Frontends: Das Frontend soll durch den Identity Provider authentifiziert werden.
- Issue #8 grafische Benutzeroberfläche für den Communication Service: Eine einfache Benutzeroberfläche soll erstellt werden. Die Ausgang-URLs des Communication Services sollen eingestellt werden können.

Alle der oben genannten Issues wurden erfolgreich implementiert.

5.4.2 Backend

Der Communication-Service wurde um API-Endpunkte erweitert, die das Hinzufügen, Aktualisieren und Löschen von Output-URLs ermöglichen. Zudem wurde der bestehende Endpunkt zur Datenübertragung so angepasst, dass die gesendeten Daten an alle aktiven und zuvor konfigurierten URLs weitergeleitet werden.

5.4.2.1 Vergleich mit den OneDev-Issues

- Issue #16: Setzen von Input- und Output-URLs: Der Communication-Service soll um Endpunkte erweitert werden, die es dem UI ermöglichen, Output-URLs zu setzen. Es soll möglich sein, mehrere URLs zu speichern, abzufragen und zu entfernen.

5.5 MongoDB-UI

5.5.1 Backend

Der Backend-Service der MongoDB-UI bietet eine REST-API, über welche man alle relevanten Informationen über die Daten in der Datenbank auslesen kann. Zu diesen Informationen gehören:

- Die Namen aller Sub-Datenbanken in der MongoDB-Datenbank. Diese können von der API unter der URL *"/DBNames"* abgefragt werden
- Die Namen aller Collections innerhalb einer Sub-Datenbank. Diese können von der API unter der URL *"/CollNames/<DBName>"* abgefragt werden
- Alle Dokumente innerhalb einer Collection. Diese können von der API unter der URL *"/Documents/<DBName>/<CollName>"* abgefragt werden

Das Backend nutzt das Authentication-Module wodurch man nur auf die API zugreifen kann, wenn ein gültiger Access Token mitgesendet wird. In Abbildung 38 ist die von Flask generierte Docs-Seite des Backends zu sehen, wo die verfügbaren API-Endpoints aufgelistet sind.

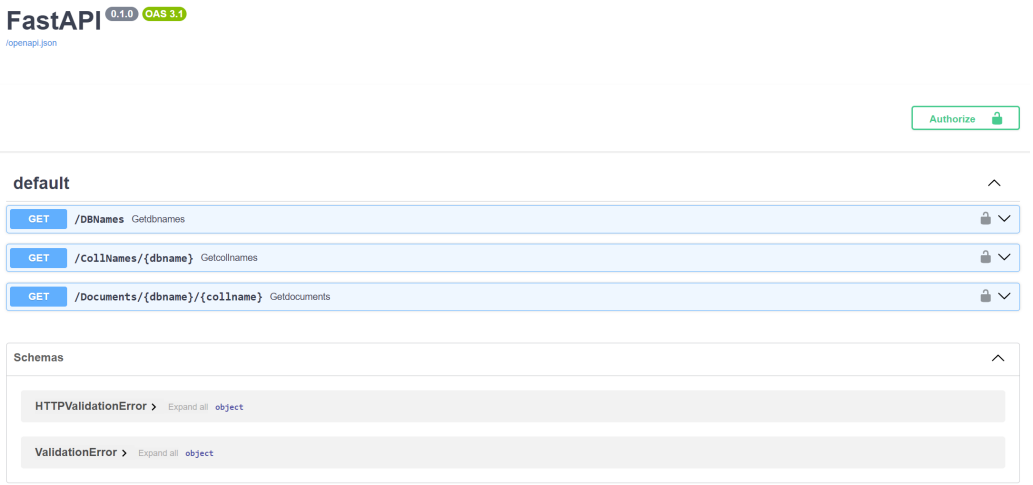


Abbildung 38: API Docs

5.5.1.1 Vergleich mit OneDev-Issues

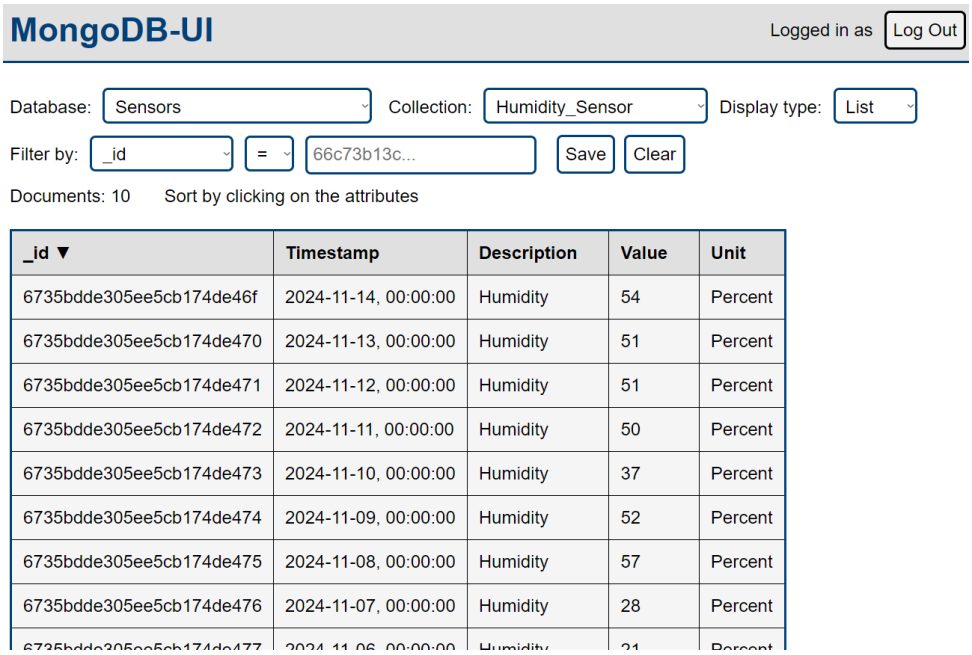
Folgende OneDev-Issues wurden für das Backend definiert:

- Issue #1 UI für MongoDB - Für die MongoDB-Datenbank soll eine UI inklusive Backend implementiert werden, welchen einen Einblick auf die Daten in der MongoDB-Datenbank liefert.
- Issue #2 Daten in der MongoDB-UI anzeigen - Es soll möglich sein die Collection, aus welcher die Dokumente angezeigt werden sollen, auszuwählen.
- Issue #23: Tests für MongoDB-UI - Alle Funktionalitäten bezüglich MongoDB-UI sollen mit Unit-Tests abgedeckt werden.
- Issue #25: Authentifizierung für MongoDB-UI Backend - Das MongoDB-UI Backend soll nur für Nutzer zugänglich sein die über einen Access Token verfügen.

Alle aufgelisteten Issues wurden abgeschlossen. Issue #1, #2 und #23 sind auch für das Frontend der MongoDB-UI relevant. Die originalen Issues sind in der Sektion B zu sehen.

5.5.2 Frontend

Das Frontend der MongoDB-UI bietet eine Webseite, welche alle Dokumente aus einer Collection und Sub-Datenbank anzeigt, welche vom Nutzer ausgewählt werden. Es bietet zwei Ansichts-Typen welche die Dokumente in Form von Kästchen und in Form von Zeilen in einer Tabelle anzeigt. Beide Ansichten sind in den Abbildungen 39 und 40 zu sehen.



_id ▼	Timestamp	Description	Value	Unit
6735bdde305ee5cb174de46f	2024-11-14, 00:00:00	Humidity	54	Percent
6735bdde305ee5cb174de470	2024-11-13, 00:00:00	Humidity	51	Percent
6735bdde305ee5cb174de471	2024-11-12, 00:00:00	Humidity	51	Percent
6735bdde305ee5cb174de472	2024-11-11, 00:00:00	Humidity	50	Percent
6735bdde305ee5cb174de473	2024-11-10, 00:00:00	Humidity	37	Percent
6735bdde305ee5cb174de474	2024-11-09, 00:00:00	Humidity	52	Percent
6735bdde305ee5cb174de475	2024-11-08, 00:00:00	Humidity	57	Percent
6735bdde305ee5cb174de476	2024-11-07, 00:00:00	Humidity	28	Percent

Abbildung 39: MongoDB-UI List View

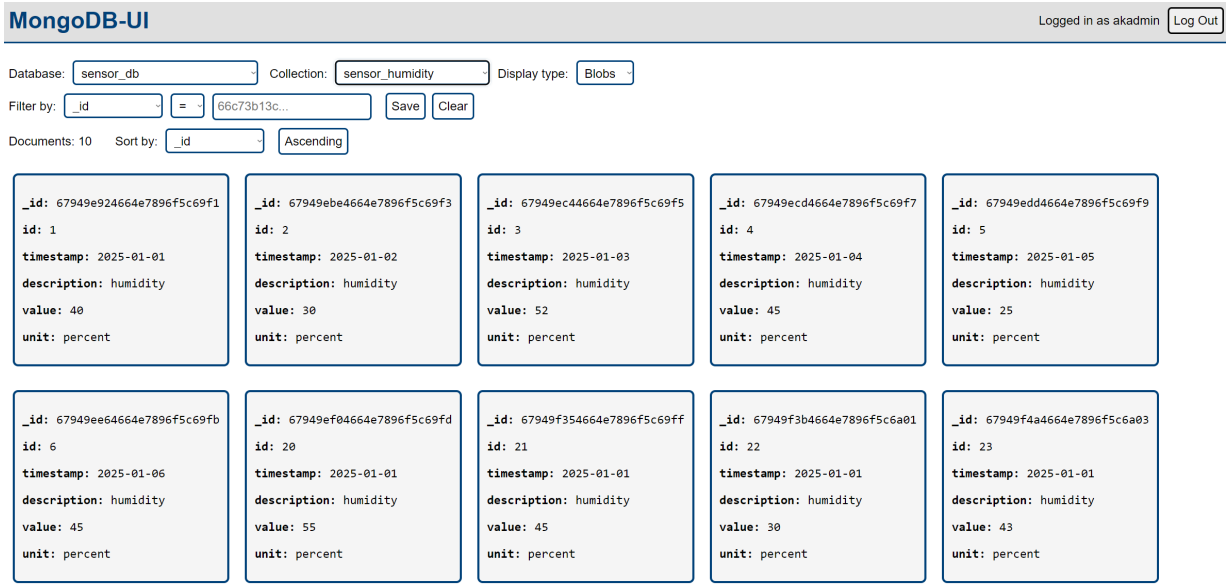


Abbildung 40: MongoDB-UI Blobs View

Die Webseite bietet außerdem Funktionalitäten für das Filtern von Daten. Hierbei ist es möglich das Attribut, nach dem gefiltert werden soll, den Vergleich und den gesuchten Wert auszuwählen. Für Attribute, welche Zahlen oder ein Datum speichern, ist ein größer/kleiner Vergleich verfügbar. Für alle Attribute ist ein "contains"Vergleich verfügbar. Es ist außerdem möglich mehrere Filter gleichzeitig aktiv zu haben. Das Filter-Menü mit mehreren aktiven Filtern und den gefilterten Tabellen ist in Abbildung 41 zu sehen.

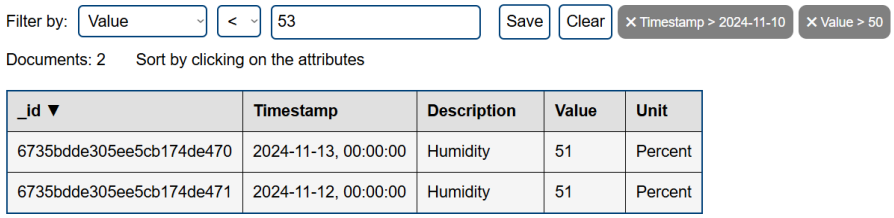


Abbildung 41: Filter-Menü mit mehreren aktiven Filtern und dazugehöriger Tabelle

Weiters ist es möglich die Dokumente zu sortieren. Sollte die List-Ansicht aktiv sein, ist das möglich, indem man auf die Header-Zellen der Tabelle klickt. Im Falle der Blobs-Ansicht erscheint ein eigenes Sortier-Menü, welches in Abbildung 40 zu sehen ist.

Die Webseite ist mittels Authentik geschützt, sollte man sich nicht bei Authentik eingeloggt haben, wird man beim Zugreifen auf die Webseite zum zentralen Login von Authentik weitergeleitet um sich dort einzuloggen. Während man nicht eingeloggt ist, wird auf der MongoDB-UI Webseite eine "You are not Logged in"Meldung angezeigt, welche in Abbildung 42 zu sehen ist.

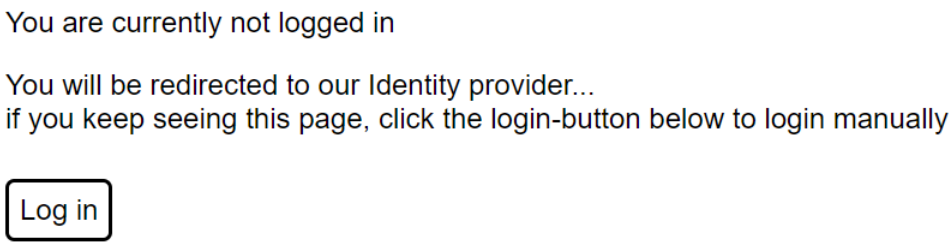


Abbildung 42: LoggedOut Ansicht der MongoDB-Webseite

5.5.2.1 Vergleich mit OneDev-Issues

Folgende OneDev-Issues wurden für das Frontend definiert:

- Issue #1 UI für MongoDB - Für die MongoDB Datenbank soll eine UI inklusive Backend implementiert werden, welchen einen Einblick auf die Daten in der MongoDB Datenbank liefert.
- Issue #2 Daten in der MongoDB-UI anzeigen - Es soll möglich sein, die Collection, aus welcher die Dokumente angezeigt werden sollen, auszuwählen.
- Issue #3 Daten in MongoDB-UI filtern - Benutzer sollen die Möglichkeit haben, Daten in der UI zu filtern.
- Issue #5 Authentifizierung für MongoDB-UI - Die Webseite soll mit einem Login in Kombination mit Authentik abgesichert werden.
- Issue #24 Daten in MongoDB-UI sortieren - Benutzer sollen die Möglichkeit haben, Daten in der UI absteigend und aufsteigend zu sortieren. In der List View sollen die Daten über das Klicken auf den Tabellen-Header sortiert werden.
- Issue #23: Tests für MongoDB-UI - Alle Funktionalitäten bezüglich MongoDB-UI sollen mit Unit-Tests abgedeckt werden.
- Issue #27: MongoDB-UI Verbesserungen - Die UI soll nutzerfreundlicher gestaltet werden, die genauen Anforderungen wurden verbal vereinbart.

Alle aufgelisteten Issues wurden abgeschlossen. Bei Issue #27 wurde das CSS überarbeitet um das Umher-rutschen von Webseiten-Elementen zu verhindern, außerdem wurde die Größe mehrerer Elemente angepasst. Die Änderungen wurden auf Basis von Nutzer-Feedback implementiert.

6 Resümee

6.1 Klaus Mühlbachler

Im Rahmen dieser Diplomarbeit habe ich vor allem viel über Containerisierung und CI/CD gelernt. Auch beim Thema Webentwicklung habe ich einiges dazugelernt, wobei ich hier auch viel bereits Gelerntes einbringen konnte, was meine Sicherheit im Gebiet Webentwicklung verstärkte. Vor allem beim Thema Deployment im CI/CD Bereich musste ich am Anfang viel lernen, da wir mit diesem Thema und den damit zusammenhängenden Technologien noch kaum gelernt hatten. Bei eben diesen Themen konnte uns unsere Betreuung im Unternehmen aber viel weiterhelfen. Das Thema Authentifizierung war in meinem Teil der Diplomarbeit nicht so schwerwiegend wie in den anderen Teilen, jedoch hab ich auch einiges über Authentifizierungsverfahren in der Webentwicklung gelernt, was ich als äußerst wichtig und interessant empfand. Bei der Entwicklung eines solchen Software-Projekts in einer Unternehmensumgebung habe ich außerdem viel über Versionsverwaltung, Issues und das Umsetzen von Feedback gelernt. Mir hat es sehr gefallen, dass wir viele Freiheiten bezüglich Design und Aufbau der Webseiten, Services und Module hatten, und auch damit, wie wir die Funktionalitäten entwickeln. Das hat in Kombination mit konstruktivem Feedback von anderen Mitarbeitern im Unternehmen dazu geführt, dass ich eine Webanwendung nach meinen Ideen entwickeln konnte, welche den Ansprüchen des Unternehmens trotzdem gerecht kommt und für Nutzer angenehm zu verwenden ist.

6.2 Christoph Amort

Ich habe diese Diplomarbeit als eine großartige Möglichkeit empfunden, mein bereits vorhandenes Wissen in der Webentwicklung mit Angular zu vertiefen und zu verfeinern. Durch diese Diplomarbeit konnte ich zusätzlich lernen, wie man Webseiten richtig mit einem Identity Provider und dem OpenID Standard authentifiziert. Gerade die Erkenntnis, wie man Benutzer, Berechtigungen und Webseiten mithilfe eines Zentralen Identity Providers verwalten und authentifizieren kann, wird mir in meiner zukünftigen Laufbahn definitiv zugutekommen.

6.3 Thomas Fragner

Im Rahmen dieser Diplomarbeit habe ich besonders viel über die Programmiersprache Python sowie den Aufbau und die Authentifizierung von APIs gelernt. Das Thema API-Entwicklung war ein zentraler Bestandteil dieser Diplomarbeit, wodurch ich ein tiefes Verständnis dafür entwickeln konnte, wie moderne Schnittstellen strukturiert und abgesichert werden. Besonders spannend fand ich die Auseinandersetzung mit verschiedenen Authentifizierungsverfahren – insbesondere mit JWT-Tokens, deren Aufbau und Einsatzmöglichkeiten ich nun deutlich besser verstehe. Auch das Thema CI/CD spielte eine große Rolle. Ich konnte praktische Erfahrungen damit sammeln, wie man automatisierte Pipelines zur Qualitätssicherung und zum Deployment einrichtet. Das war anfangs herausfordernd, hat mir aber einen wertvollen Einblick in moderne Softwareentwicklung gegeben. In diesem Zusammenhang habe ich auch intensiv mit Docker gearbeitet, wodurch ich gelernt habe, wie man Anwendungen containerisiert und für verschiedene Umgebungen vorbereitet. Ein weiterer wichtiger Punkt war der Umgang mit Git, insbesondere mit Git-Submodules, was mir half, größere Projekte modular und übersichtlich zu strukturieren. Die Arbeit mit Versionsverwaltung, das Verwalten von Issues und die Umsetzung von Feedback aus dem Team haben meine Arbeitsweise stark verbessert.

6.4 Tobias Ganglberger

Mithilfe dieser Diplomarbeit ist es mir gelungen, viel über die Programmiersprache Python zu lernen, insbesondere über die Implementierung von REST-APIs, und wie man diese richtig absichert. Die größte Herausforderung war für mich anfangs der Umgang mit Kubernetes-Cluster und CI/CD, da ich in diesen Bereichen kaum Erfahrung hatte, im Rahmen dieser Diplomarbeit ist es mir aber gelungen, mich einzufinden und gute Einblicke in diese Bereiche zu bekommen. Ein weiterer für mich persönlich spannender Aspekt war für mich die Organisation für die Implementierung dieser Diplomarbeit. Ich habe gelernt, dass die Arbeit mit Organisationstechniken wie Versionierung, Issues und Git-Branches in einer professionellen Umgebung unabdingbar sind, um ein effizientes Arbeiten zu ermöglichen. Meine in dieser Diplomarbeit gesammelten Erfahrungen werden für mich in meinem Zukünftigen Leben bestimmt von großem Nutzen sein.

Glossar

API Application Programming Interface; Eine Programmier-Schnittstelle, welche von Programmen genutzt wird um miteinander zu kommunizieren und Daten auszutauschen [82]

CI/CD Continuous Integration / Continues Deployment; Ein Vorgehen bei dem Applikationen während der Entwicklung automatisch getestet und deployed werden.[83]

CORS Cross-Origin Resource Sharing; Eine Regel, welche im Web verwendet wird um zu verbieten, dass eine Webseite Daten von externen Webseiten läd

JSON JavaScript Object Notation; Ein sprachunabhängiges, offenes und für Menschen lesbares Dateiformat, welches für den Austausch von Daten verwendet wird.[36]

Kubernetes Secret Eine Kubernetes-Ressource zur sicheren Speicherung sensibler Daten wie Passwörter, Tokens oder URLs. [72]

MFA Mehr Faktor Authentifizierung

UI User-Interface / Benutzeroberfläche

Literaturverzeichnis

[1] OpenID Foundation, „About OpenID," 2024, Letzter Zugriff am 26.03.2025. Online verfügbar: https://openid.net/developers/discover-openid-and-openid-connect/

[2] G. Developers, „Understanding OpenID and OAuth," 2024, Letzter Zugriff am 26.03.2025. Online verfügbar: https://developers.google.com/identity/protocols/oauth2

[3] D. Hardt, „The OAuth 2.0 Authorization Framework," 2012, RFC 6749, Letzter Zugriff am 26.03.2025. Online verfügbar: https://datatracker.ietf.org/doc/html/rfc6749

[4] Wikipedia, „OpenID – Wikipedia," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://de.wikipedia.org/wiki/OpenID

[5] I. Talend, „REST-API: Definition, Funktionen und Bedingungen," 2025, letzter Zugriff am 12.03.2025. Online verfügbar: https://www.talend.com/de/resources/was-ist-rest-api/

[6] A. S. Inc, „Authentik Docs," 2025, letzter Zugriff am 24.03.2025. Online verfügbar: https://docs.goauthentik.io/docs/

[7] I. SE, „Was ist Single-Sign-On?" 2025, letzter Zugriff am 24.03.2025. Online verfügbar: https://www.ionos.at/digitalguide/server/tools/single-sign-on/

[8] Wikipedia, „Docker (Software)," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://de.wikipedia.org/wiki/Docker_(Software)

[9] Microsoft, „Container im Vergleich zu virtuellen Computern," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://learn.microsoft.com/de-de/virtualization/windowscontainers/about/containers-vs-vm

[10] T. K. Authors, „Overview," 2024, Letzter Zugriff am 28.03.2025. Online verfügbar: https://kubernetes.io/docs/concepts/overview/

[11] ——, „CronJob," 2024, Letzter Zugriff am 28.03.2025. Online verfügbar: https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

[12] P. S. Foundation, „Welcome to Python.org," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://www.python.org/

[13] ——, „The Python Package Index (PyPI)," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://pypi.org/

[14] T. Peters, „The Zen of Python," 2004, PEP 20, Letzter Zugriff am 26.03.2025. Online verfügbar: https://peps.python.org/pep-0020/

[15] S. Ramírez, „FastAPI Documentation," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://fastapi.tiangolo.com/

[16] ——, „Path Operation Decorators - FastAPI," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://fastapi.tiangolo.com/tutorial/path-params/

[17] ——, „Concurrency and async / await - FastAPI," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://fastapi.tiangolo.com/async/

[18] M. Inc., *MongoDB Manual: PyMongo Documentation*, 2024, letzter Zugriff am 11.03.2025. Online verfügbar: https://www.mongodb.com/docs/languages/python/pymongo-driver/current/

[19] ——, *MongoDB Manual: PyMongo Documentation, Connect to Database*, 2024, letzter Zugriff am 11.03.2025. Online verfügbar: https://www.mongodb.com/docs/languages/python/pymongo-driver/current/connect/

[20] ——, *MongoDB Manual: PyMongo Documentation, Access Databases and Collections*, 2024, letzter Zugriff am 11.03.2025. Online verfügbar: https://www.mongodb.com/docs/languages/python/pymongo-driver/current/databases-collections/

[21] A. Sulcas, „Python Requests Library: 2025 Guide," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://oxylabs.io/blog/python-requests

[22] Wikipedia, „Requests (software)," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://en.wikipedia.org/wiki/Requests_(software)

[23] D. Radavicius, „HTTPX vs Requests vs AIOHTTP," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://oxylabs.io/blog/httpx-vs-requests-vs-aiohttp

[24] „HTTPX," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://www.python-httpx.org/

[25] E. O. Ltd., „Uvicorn Documentation," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://www.uvicorn.org/

[26] ——, „Uvicorn: ASGI Server Implementation," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://www.uvicorn.org/#performance

[27] Parkayun, „bson 0.5.10," 2020, Letzter Zugriff am 26.03.2025. Online verfügbar: https://pypi.org/project/bson/

[28] holger krekel und pytest-dev team, *pytest documentation: history*, 2015, letzter Zugriff am 26.03.2025. Online verfügbar: https://www.mongodb.com/resources/languages/bson

[29] nikhilaggarwal3, „Python datetime module," 2023, Letzter Zugriff am 26.03.2025. Online verfügbar: https://www.geeksforgeeks.org/python-datetime-module/?ref=gcse_outind

[30] „kubernetes.client.CoreV1Api," 2024, Letzter Zugriff am 26.03.2025. Online verfügbar: https://github.com/kubernetes-client/python/blob/master/kubernetes/docs/CoreV1Api.md

[31] „kubernetes.client.BatchV1Api," 2024, Letzter Zugriff am 26.03.2025. Online verfügbar: https://github.com/kubernetes-client/python/blob/master/kubernetes/docs/BatchV1Api.md

[32] T. K. Authors, „Client Libraries," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://kubernetes.io/docs/reference/using-api/client-libraries/

[33] D. Choudhary, „Unleashing the Power of UUID in Python: A Comprehensive Guide," 2024, Letzter Zugriff am 26.03.2025. Online verfügbar: https://blog.devops.dev/unleashing-the-power-of-uuid-in-python-a-comprehensive-guide-440a42d7b520

[34] P. S. Foundation, „base64 — Base16, Base32, Base64, Base85 Data Encodings," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://docs.python.org/3/library/base64.html

[35] M. Nealer, „A Practical Guide to using Pydantic," 2024, Letzter Zugriff am 26.03.2025. Online verfügbar: https://medium.com/@marcnealer/a-practical-guide-to-using-pydantic-8aafa7feebf6

[36] inray Industriesoftware GmbH, „Was ist JSON?" 2025, letzter Zugriff am 23.03.2025. Online verfügbar: https://www.opc-router.de/was-ist-json/

[37] P. S. Foundation, „json — JSON encoder and decoder," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://docs.python.org/3/library/base64.html

[38] holger krekel und pytest-dev team, *pytest documentation*, 2015, letzter Zugriff am 11.03.2025. Online verfügbar: https://docs.pytest.org/en/stable/

[39] ——, *pytest documentation: history*, 2015, letzter Zugriff am 11.03.2025. Online verfügbar: https://docs.pytest.org/en/stable/history.html

[40] M. Inc. (2025) MongoDB Logo. letzter Zugriff am 11.03.2025. Online verfügbar: https://www.mongodb.com/company/newsroom/brand-resources

[41] ——, *MongoDB Manual: Introduction*, 2024, letzter Zugriff am 06.03.2025. Online verfügbar: https://www.mongodb.com/docs/manual/introduction/

[42] T. Joes, „A Brief History of MongoDB," *tutorjoes*, 2023, letzter Zugriff am 06.03.2025. Online verfügbar: https://www.tutorjoes.in/mongodb_tutorial/history_of_mongodb

[43] M. Inc., *MongoDB Manual: Databases and Collections*, 2024, letzter Zugriff am 06.03.2025. Online verfügbar: https://www.mongodb.com/docs/manual/core/databases-and-collections/

[44] ——, *MongoDB Manual: CRUD Operations*, 2024, letzter Zugriff am 06.03.2025. Online verfügbar: https://www.mongodb.com/docs/manual/crud/

[45] ——, *MongoDB Manual: Aggregation Operations*, 2024, letzter Zugriff am 06.03.2025. Online verfügbar: https://www.mongodb.com/docs/manual/aggregation/

[46] „Wikipedia: Angular full color logo," 2024, letzter Zugriff am 08.03.2025. Online verfügbar: https://upload.wikimedia.org/wikipedia/commons/thumb/c/cf/Angular_full_color_logo.svg/512px-Angular_full_color_logo.svg.png

[47] S. GMBH, „Was ist Angular," 2025, letzter Zugriff am 24.03.2025. Online verfügbar: https://www.studysmarter.de/schule/informatik/webentwicklung/angular/

[48] G. LLC, „Angular Componets," 2025, letzter Zugriff am 24.03.2025. Online verfügbar: https://v17.angular.io/guide/component-overview

[49] I. SE, „Single Page Applications," 2025, letzter Zugriff am 24.03.2025. Online verfügbar: https://www.ionos.at/digitalguide/websites/webseiten-erstellen/single-page-application/

[50] O. Foundation, „What is NodeJS," 2025, letzter Zugriff am 24.03.2025. Online verfügbar: https://nodejs.org/en/about

[51] ——, „An introduction to the npm package manager," 2025, letzter Zugriff am 24.03.2025. Online verfügbar: https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager

[52] Microsoft, „TypeScript is JavaScript with syntax for types." 2025, letzter Zugriff am 24.03.2025. Online verfügbar: https://www.typescriptlang.org/

[53] NPM, „angular-oauth2-oidc," 2025, letzter Zugriff am 24.03.2025. Online verfügbar: https://www.npmjs.com/package/angular-oauth2-oidc

[54] I. NGINX, „Offizielle nginx Webseite," 2025, letzter Zugriff am 12.03.2025. Online verfügbar: https://nginx.org

[55] K. Inc., „Was ist Nginx? Ein grundlegender Blick darauf, was es ist und wie es funktioniert," 2023, letzter Zugriff am 12.03.2025. Online verfügbar: https://kinsta.com/de/wissensdatenbank/was-ist-nginx/

[56] O. Pty, „DevOps Platform Powerful. Simple." 2023-2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://onedev.io/

[57] S. LLC, „What is Rancher?" 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://ranchermanager.docs.rancher.com/

[58] ——, „Why Rancher?" 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://www.rancher.com/why-rancher

[59] I. Coder Technologies, „Deploy development environments on your infrastructure." 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://coder.com/

[60] ——, „Workspace Management," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://coder.com/docs/user-guides/workspace-management

[61] ——, „Our Team," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://coder.com/about/

[62] G. Inc., „What is CI/CD?" 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://about.gitlab.com/topics/ci-cd/

[63] O. Team, „OneDev Documentation – CI/CD Guide," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://code.onedev.io/projects/1/content/readme

[64] T. K. Authors, „kubectl CLI Tool - Kubernetes," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://kubernetes.io/docs/reference/kubectl/

[65] S. Chacon und B. Straub, „Git Tools - Submodules," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://git-scm.com/book/en/v2/Git-Tools-Submodules

[66] P. P. Authority, „Requirements Files – pip documentation," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://pip.pypa.io/en/stable/user_guide/#requirements-files

[67] Auth0, „JWT.IO Introduction," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://jwt.io/introduction

[68] P. Maintainers, „PyJWT Documentation," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://pyjwt.readthedocs.io/en/stable/

[69] E. O. Ltd., „httpx - A next generation HTTP client for Python," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://www.python-httpx.org/

[70] D. Inc., „Environment variables in Docker," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://docs.docker.com/compose/environment-variables/

[71] S. Ramírez, „Security - FastAPI Documentation," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://fastapi.tiangolo.com/tutorial/security/oauth2-jwt/

[72] T. K. Authors, „Managing Secrets," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://kubernetes.io/docs/concepts/configuration/secret/

[73] ——, „Python client for Kubernetes," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://github.com/kubernetes-client/python

[74] ——, „Managing Secrets using kubectl," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://kubernetes.io/docs/tasks/configmap-secret/managing-secret-using-kubectl/

[75] M. D. Network, „HTTP response status codes," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

[76] S. LLC, „Rancher Authentication and RBAC," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://ranchermanager.docs.rancher.com/api/api-tokens/

[77] Bootstrap, „Bootstrap Docs," 2025, letzter Zugriff am 24.03.2025. Online verfügbar: https://getbootstrap.com/docs/5.3/getting-started/download/

[78] Microsoft, „API design guidance," 2025, Letzter Zugriff am 26.03.2025. Online verfügbar: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design

[79] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures," 2000, Letzter Zugriff am 26.03.2025. Online verfügbar: https://restfulapi.net/http-methods/

[80] G. LLC, „Angular Material," 2025, letzter Zugriff am 24.03.2025. Online verfügbar: https://material.angular.io/guide/getting-started

[81] M. Inc., *Aggregation Operators*, 2024, letzter Zugriff am 27.03.2025. Online verfügbar: https://www.mongodb.com/docs/manual/reference/operator/aggregation/

[82] L. GmbH, „Was ist eine API?" letzter Zugriff am 26.03.2025. Online verfügbar: https://www.locoia.com/de/api-definition-und-funktion/

[83] I. RedHat, „What is CI/CD?" 2025, letzter Zugriff am 23.03.2025. Online verfügbar: https://www.redhat.com/de/topics/devops/what-is-ci-cd

Abbildungsverzeichnis

Anhang

A Aufgabenverteilung

A.1 Klaus Mühlbachler

Abstract
Danksagung
1.1 Einleitung - Problemstellung
1.3.2 Einleitung - Projektumfeld - Betreuung
2.1.2 Grundlagen und Methoden - Standards - REST-API
2.2.5 Grundlagen und Methoden - Technologien - MongoDB
2.2.4.2 Grundlagen und Methoden - Technologien - Python - pymongo
2.2.4.13 Grundlagen und Methoden - Technologien - Python - pytest
2.2.7 Grundlagen und Methoden - Technologien - Python - nginx
3.1 Konzept - Überblick
3.11 Konzept - Frontend-Authentifizierung
3.9 Konzept - MongoDB-UI
4.1.2 Implementierung - Deployment - Dockerfiles
4.4 Implementierung - Fast-API Anwendungen
4.7 Implementierung - MongoDB-UI
5.5 Ergebnis - MongoDB-UI
6.1 Resümee - Klaus Mühlbachler

Im praktischen Teil der Diplomarbeit hat Klaus Mühlbachler das Frontend und Backend der MongoDB-UI implementiert.

A.2 Christoph Amort

1.3.1 Einleitung – Projektumfeld – Projektteam
1.3.3 Einleitung – Projektumfeld – Auftraggeber
2.2.1 Grundlagen und Methoden – Technologien – Authentik
2.2.6 Grundlagen und Methoden – Technologien – Angular
3.3 Konzept – Screening-Service-UI
3.5 Konzept – Communication-Service-UI
3.10 Konzept – Identity Provider
4.2 Implementierung – Authentik
4.5.1 Implementierung – Communication-Service – Frontend
4.6.1 Implementierung – Screening-Service – Frontend
5.1 Ergebnis – Authentik
5.3.1 Ergebnis – Screening-Service – Frontend
5.4.1 Ergebnis – Communication-Service – Frontend
6.2 Resümee – Christoph Amort

Im praktischen Teil der Diplomarbeit hat Christoph Amort das Frontend für den Screening Srevice und den Communication Service implementiert sowie den Identity Provider Authentik aufgesetzt.

A.3 Tobias Ganglberger

Kurzfassung
1.2 Einleitung - Zielsetzung
2.2.2 Grundlagen und Methoden - Technologien - Docker
2.2.3 Grundlagen und Methoden - Technologien - Kubernetes
2.2.4.3 Grundlagen und Methoden - Technologien - Python - requests
2.2.4.4 Grundlagen und Methoden - Technologien - Python - httpx
2.2.4.6 Grundlagen und Methoden - Technologien - Python - bson
2.2.4.7 Grundlagen und Methoden - Technologien - Python - datetime
2.2.4.8 Grundlagen und Methoden - Technologien - Python - kubernetes
2.2.4.9 Grundlagen und Methoden - Technologien - Python - uuid
2.2.4.10 Grundlagen und Methoden - Technologien - Python - base64
2.2.4.11 Grundlagen und Methoden - Technologien - Python - pydantic
2.2.4.12 Grundlagen und Methoden - Technologien - Python - json
2.3.1 Grundlagen und Methoden - Tools - OneDev
2.3.3 Grundlagen und Methoden - Tools - Coder
3.1.1 Konzept - Überblick - Beispiel-Ablauf im fertigen System
3.4 Konzept - Screening-Service-Erweiterung
3.8 Konzept - Database-Module
4.3.5 Implementierung - Modules - Database Module
4.6.2 Implementierung - Screening-Service - Backend
4.5.2.3 Implementierung - Communication-Service - Backend - Tests
5.2.3 Ergebnis - Modules - Database-Module
5.3.2 Ergebnis - Screening-Service - Backend
6.4 Resümee - Tobias Ganglberger

Im praktischen Teil der Diplomarbeit hat Tobias Ganglberger das Backend für den Screening Srevice und das Database-Module implementiert. Darüber hinaus hat er dem Cronjob die Funktionalitäten für die Verwaltung der Berechnungsmethoden und eine neue Berechnungsmethode implementiert.

A.4 Thomas Fragner

2.1.1 OpenID
2.2.4 Python
2.2.4.1 FastAPI
2.3.2 Rancher
3.6 Konzept - Communication Service Backend Erweiterung
3.7 Konzept - Secret Communication Module
3.12 Konzept- Authentication Module
4.1.1 Implementierung - CI/CD
4.1.1.1 Implementierung - CI/CD - OneDev BuildSpec
4.1.1.2 Implementierung - CI/CD - Kubernetes File
4.3.1 Implementierung - Module - Einbindung von Modulen als Submodul
4.3.2 Implementierung - Authentication Module
4.3.3 Implementierung - Secret Communication Module
4.5.2 Implementierung - Communication Service - Backend

4.5.2.1 Implementierung - Communication Service - Backend - Vorhandene Endpoints
4.5.2.2 Implementierung - Communication Service - Backend - Neue Endpoints
5.2.1 Ergebnis - Module - Authentication Module
5.2.2 Ergebnis - Module - Secret Communication Module
5.4.2 Ergebnis - Communication Service - Backend
6.3 Resümee - Thomas Fragner

Im praktischen Teil seiner Diplomarbeit erweiterte Thomas Fragner das Backend des Communication Services um zusätzliche Endpoints, die das Erstellen, Bearbeiten und Löschen von Communication-Service-URLs ermöglichen. Darüber hinaus implementierte er die Authentifizierung sämtlicher Python-Services sowie die sichere Kommunikation mit Kubernetes-Secrets.

B Meilensteine

Meilensteine für dieses Software-Projekt wurden als Liste von OneDev Issues definiert. Die Meilensteine wurden in Englisch verfasst, und lauten wie folgt:

- #1 UI for MongoDB: A Simple UI shall be created, that connects to MongoDB and displays the connection status. (connected or disconnected)
- #2 Show Data for MongoDB: The UI of the MongoDB shall be extended by the possibility to show different collections of the databse. There shall be the possibility to navigate between the collections.
- #3 Filtering of MongoDB data in UI: The user shall have the possibility to filter the data of the MongoDB in the UI.
- #4 Authentik Setup: Authentik shall be set up to handle the authentication of other applications. The Authentik UI shall be configured appropriately and the MongoDB UI, the Communication Service UI, the Screening Service UI and the RabbitMQ UI shall be integrated.
- #5 Authentication for MongoDB UI: The UI of the MongoDB needs to have a Login Page, where the User authenticates via OpenID Connect and the Authentik deployment. No content shall be accessible without logging in.
- #6 Roles for Authentik: Authentik shall be configured with two roles: admin and user. The admin shall have all possible rights and shall be able to configure which user has access to which application. The user shall only see and be able to authenticate to applications they are authorized to.
- #7 Authentication for Screening and Communication Service UI: The UI of the Screening and Communication Service needs to have a Login Page, where the User authenticates via OpenID Connect and the Authentik deployment. No content shall be accessible without logging in.
- #9 UI for Screening-Service: A simple UI shall be created that allows the user to set the screening-service Interval and allows executing the Cronjob immediately
- #10 Additional Calculation in screening service: The screening service shall additionally calculate something new from the sensor data of the raspberry pi. You are free to choose an appropriate calculation. Therefore the communication service shall save the new sensor data to the MongoDB. The screening service shall do the calculation with the data from the MongoDB and pass the result the communication service and back to the MongoDB. The communication service shall forward the result to the Fabasoft Approve VDE.

- #11 Produce data with Raspberry Pi: The raspberry pi shall be configured as OPC-UA client and shall send sensor data to the communication service. The code of the middleware shall be updated, so that it can process new data.
- #12 Extensive calculation in screening service: The screening service shall additionally calculate something new. The data needs to be analysed beforehand and something meaningful shall be calculated. The calculation shall not be trivial. The result shall be passed to the communication service and afterwards shall be forwarded to Fabasoft Approve VDE.
- #13 Selection of calculation method: The user shall have the possibility to select the calculation method in the screening service UI.
- #14 Configure Database URL in UI: The user shall have the possibility to configure the URL to the database in the screening service UI and the communication service ui.
- #15 Authentification for Screening and Communication Service Backend: The backend APIs of the Screening and Communication Service needs to use OpenID connect and Authentik to authorize users. No content shall be accessible without logging in.
- #16 Set input and output URL endpoints: The backend Communication service shall be extended by endpoints that make it possible for the UI to set the output URLs. Currently, information messages are sent to https://vm7.vde.fabasoft.com/. It shall be possible to specify multiple URLs, to query the current URLs and to remove URLs.
- #17 Set screening interval: The screening service backend shall be extended by an endpoint, that allows to set the screening interval and also allows to execute the cronjob immediately
- #18 Set calculation method: An endpoint shall be added to the screening service, that allows to set the calculation method in the screening service.
- #19 Configure Database URL endpoint: An endpoint shall be added to the screening service and communication service backend, that is able to set the database URL.
- #22 Tests for processing new sensor data: The new data pipeline and calculation from #10 shall be tested properly with unit tests. Additionally, the existing tests may be extended.
- #23 Tests for MongoDB UI: All functionalities related to the MongoDB-UI shall be covered with UnitTests.
- #24 Sort data in MongoDB UI: The user shall have the possibility to sort the data absed on a selected column in the MongoDB list vie. Sorting shall be possible in ascending and descending order. Preferably sorting the table shall be possible by clicking on the respective column.
- #25 Authentication for MongoDB backend: The MongoDB backend shall not be accessible for users not logged in, in the Authentik deployment
- #26 Authentication for Screening service backend: The screening service backend API shall not be accessible for users not logged in, in the Authentik deployment.
- #27 MongoDB UI enhancements: The MongoDB ZU shall be made more user-friendly. The necessary changes were discussed verbally.

Manche Issues wurden nach der Definition verbal abgeändert.

C Dateien

C.1 Muster-Dockerfile Backend

```
FROM docker/python:3.9-slim
# Umgebungsvariablen für Proxy-Einstellungen
ENV HTTP_PROXY "http://fabaproxylnz.fabagl.fabasoft.com:8080"
ENV HTTPS_PROXY "http://fabaproxylnz.fabagl.fabasoft.com:8080"
ENV http_proxy "http://fabaproxylnz.fabagl.fabasoft.com:8080"
ENV https_proxy "http://fabaproxylnz.fabagl.fabasoft.com:8080"
ENV no_proxy
".res.fabagl.fabasoft.com,localhost,127.0.0.1,*.mindbreeze.com,192.168.*,10.
↪  *,172.16.0.0-172.31.255.255,172.19.0.1,*.fabagl.fabasoft.com"
ENV CI=true

# Umgebungsvariablen für Zugriff auf Identity Provider
ENV oidc_aud "zsPSIHku0Lp6QXAOyJNfoKjR0DC4XuA2ZykiKBYD"
ENV jwks_url
"https://demo.duendesoftware.com/.well-known/openid-configuration/jwks"
ENV oidc_url
"https://demo.duendesoftware.com/.well-known/openid-configuration/"
ENV token_url "https://demo.duendesoftware.com/connect/token"
ENV auth_url "https://demo.duendesoftware.com/connect/authorize"

# Service-spezifische Umgebungsvariablen
ENV mdbURL="mongodb://host.docker.internal:27017/"

# Arbeitsverzeichnis setzen
WORKDIR /app

# Kopieren von Dateien
COPY requirements.txt requirements.txt
COPY ./libraries /srv/libraries
COPY . .

# Bibliotheken installieren
RUN pip install --no-cache-dir -r requirements.txt

# Port freigeben
EXPOSE 5000

# Service starten
CMD ["python", "app.py"]
# Alternativ:
# CMD ["python3", "-m", "uvicorn", "src.api:app", "--host", "0.0.0.0",
↪  "--port", "8000"]
```

C.2 Muster-Dockerfile Frontend

```
FROM docker/node:18.19.0 AS build

# Proxy Einstellungen
ENV HTTP_PROXY "http://fabaproxylnz.fabagl.fabasoft.com:8080"
ENV HTTPS_PROXY "http://fabaproxylnz.fabagl.fabasoft.com:8080"
ENV http_proxy "http://fabaproxylnz.fabagl.fabasoft.com:8080"
ENV https_proxy "http://fabaproxylnz.fabagl.fabasoft.com:8080"
ENV no_proxy
↪  ".res.fabagl.fabasoft.com,localhost,127.0.0.1,*.mindbreeze.com,192.168.*
↪  ,10.*,172.16.0.0-172.31.255.255,172.19.0.1,*.fabagl.fabasoft.com"

# Arbeitsverzeichnis setzen
WORKDIR /app

# Bibliotheken installieren
COPY package*.json ./
RUN npm install
RUN npm install -g @angular/cli

COPY . ./

# Webseiten-Dateien builden
RUN ng build --configuration=production

# Hosting-Container
FROM docker/nginx:latest

# Anwendungsspezifische Umgebungsvariablen
ENV API_URL="http://127.0.0.1:5000/"
ENV issuer="https://demo.duendesoftware.com"
ENV clientId="interactive.public"

# Build-Dateien kopieren
COPY --from=build app/dist/mdb-ui/browser /usr/share/nginx/html

# Entrypoint.sh kopieren
COPY entrypoint.sh /usr/local/bin/entrypoint.sh
RUN chmod +x /usr/local/bin/entrypoint.sh

EXPOSE 80

ENTRYPOINT ["/bin/sh", "-c", "/usr/local/bin/entrypoint.sh"]
```

C.3 Logo

Authentigrade

Abbildung 43: Authentigrade Logo

C.4 Diplomarbeitsplakat

Abbildung 44: Authentigrade Diplomarbeitsplakat